ISO/IEC JTC 1/SC 32 N 2593

Date: 2015-01-25

REPLACES: 32N24638

ISO/IEC JTC 1/SC 32

Data Management and Interchange

Secretariat: United States of America (ANSI)
Administered by Farance Inc. on behalf of ANSI

DOCUMENT TYPE	Text for DIS ballot
TITLE	ISO/IEC DIS 13249-3 Information technology - Database languages - SQL Multimedia and Application Packages Part 3: Spatial 5th Edition
SOURCE	WG4 - Paul Scarponcini - project editor
PROJECT NUMBER	1.32.04.05.03.00
STATUS	text for ballot DIS 13249-3 ed 5. Disposition of comments on CD N2538 is in N2594. This text is to be sent to ITTF for DIS ballot.
REFERENCES	
ACTION ID.	ITTF
REQUESTED ACTION	
DUE DATE	--
Number of Pages	1353
LANGUAGE USED	English
DISTRIBUTION	P & L Members SC Chair WG Conveners and Secretaries

Dr. Timothy Schoechle, Secretary, ISO/IEC JTC 1/SC 32
Farance Inc *, 3066 Sixth Street, Boulder, CO, United States of America
Telephone: +1 303-443-5490; E-mail: Timothy@Schoechle.org
available from the JTC 1/SC 32 WebSite http://www.jtc1sc32.org/
*Farance Inc. administers the ISO/IEC JTC 1/SC 32 Secretariat on behalf of ANSI

ISO/IEC JTC 1/SC 32/WG 4
Secretariat: U.S.A.

ISO/IEC JTC 1/SC 32 N 2593
ISO/IEC 13249-3:201x(E)
Date: 2015-01-23

INTERNATIONAL STANDARD **ISO/IEC 13249-3**

Fifth edition
201x-mm-dd

Text for DIS Ballot

Information technology — Database languages — SQL Multimedia and application packages —

Part 3: Spatial

Technologies de l'information – Languages de bases de donées – Multimédia SQL et paquetages d'application –

Partie 3: Spatial

Document type: International Standard
Document subtype: Not applicable
Document stage: **(40) Enquiry**
Document language: E

Reference Number
ISO/IEC 13249-3:201x(E)

Printed on: January 23, 2015 19:33 EST / Version 2

PDF disclaimer

ISO copyright office
Case postale 56 • CH-1211 Geneva 20 • Switzerland
Tel. + 41 22 749 01 11
Fax +41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Canada

Contents

Page

Figures — Page

Tables Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 13249 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 13249-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This fifth edition cancels and replaces the fourth edition (ISO/IEC 13249—3:2011), which has been technically revised.

ISO/IEC 13249 consists of the following parts, under the general title *Information technology — Database languages — SQL multimedia and application packages*:

— *Part 1: Framework*

— *Part 2: Full-Text*

— *Part 3: Spatial*

— *Part 5: Still Image*

— *Part 6: Data Mining*

— *Part 7: History*

Annexes A to E and the Bibliography of this part of ISO/IEC 13249 are for information only.

Introduction

ISO/IEC 13249 defines multimedia and application-specific types and their associated routines using the features for the creation of user-defined types specified in ISO/IEC 9075, “Information technology - Database languages – SQL”.

The organization of this part of ISO/IEC 13249 is as follows:

Clause 1, "Scope", specifies the scope of this part of ISO/IEC 13249.

Clause 2, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 13249, constitute provisions of this part of ISO/IEC 13249.

Clause 3, "Terms and definitions, notations, and conventions", defines the notations and conventions used in this part of ISO/IEC 13249.

Clause 4, "Concepts", presents concepts used in the definition of this part of ISO/IEC 13249.

Clause 5, "Geometry Types", defines the geometry supertype.

Clause 6, "Point Types", defines primitive 0-dimensional geometry types.

Clause 7, "Curve Types", defines primitive 1-dimensional geometry types.

Clause 8, "Surface Types", defines primitive 2-dimensional geometry types.

Clause 9, "Solid Types", defines primitive 3-dimensional geometry types.

Clause 10, "Geometry Collection Types", defines the geometry collection types.

Clause 11, "Topology-Geometry", defines node, edge, and face topology-geometry primitives.

Clause 12, "Topology-Network", defines node and link topology-network primitives.

Clause 13, "General Routines", defines the routines to determine shortest path in directed or undirected graphs.

Clause 14, "Spatial Reference System Type", defines the user-defined type to manage spatial reference systems.

Clause 15, "Linear Referencing Types", defines user-defined types to manage linear referencing.

Clause 16, "Angle and Direction Types", defines the angles and direction types.

Clause 17, "Support Types", defines supporting types and routines used by this part of ISO/IEC 13249.

Clause 18, "Support Routines", defines supporting functions and procedures used by this part of ISO/IEC 13249.

Clause 19, "SQL/MM Spatial Information Schema" defines the SQL/MM Spatial Information Schema.

Clause 20, "SQL/MM Spatial Definition Schema" defines the SQL/MM Spatial Definition Schema.

Clause 21, "SQL/MM Linear Referencing Information and Definition Schemas" defines the SQL/MM Linear Referencing Information and Definition Schemas.

Clause 22, "Status Codes", defines the SQLSTATE codes used in this part of ISO/IEC 13249.

Clause 23, "Conformance", defines the criteria for conformance to this part of ISO/IEC 13249.

Annex A, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 13249 states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementer shall provide in each case.

Annex B, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 13249 states explicitly that the meaning or effect on the database is implementation-dependent.

Annex C, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this part of ISO/IEC 13249.

Annex D, "Incompatibilities with ISO/IEC 13249-3:2006", is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 13249-3.

Annex E, "Geometry Type Hierarchy", is an informative Annex. It visually describes the inheritance relationship between user-defined types in this part of ISO/IEC 13249.

Bibliography is the last informative Annex. It is a list of selective reading relating to this part of ISO/IEC 13249.

In the text of this part of ISO/IEC 13249, in Clause 5, "Geometry Types", through Clause 20, "SQL/MM Spatial Definition Schema", subclauses begin on a new page. Any resulting blank space is not significant.

The spatial user-defined types defined in this part adhere to the following:

— A spatial user-defined type is generic to spatial data handling. It addresses the need to store, manage and retrieve information based on aspects of spatial data such as geometry, location, and topology.

— A spatial user-defined type does not redefine the database language SQL directly or in combination with another spatial data type.

Implementations of this part of ISO/IEC 13249 may exist in environments that also support geographic information, decision support, data mining, and data warehousing systems.

Application areas addressed by implementations of this part of ISO/IEC 13249 include, but are not restricted to, automated mapping, desktop mapping, facilities management, geoengineering, graphics, linear referencing, location based services, multimedia, and resource management applications.

Blank page

Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial

1 Scope

This part of ISO/IEC 13249:

a) defines concepts specific to this part of ISO/IEC 13249,

b) defines spatial user-defined types and their associated routines.

2 Normative references

The following referenced documents are indispensable for the application of this part of ISO/IEC 13249. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9075-1, Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).

ISO/IEC 9075-2, Information technology — Database languages — SQL — Part 1: Foundation (SQL/Foundation).

ISO/IEC 13249-1, *Information technology — Database languages — SQL multimedia and application packages — Part 1: Framework*.

ISO 19107, *Geographic information — Spatial schema*.

ISO 19111, *Geographic information — Spatial referencing by coordinates*.

ISO 19136, *Geographic information — Geography Markup Language*.

ISO 19136-2, *Geographic information — Geography Markup Language - Extended schemas and encoding rules*.

ISO 19148, *Geographic information — Linear referencing*

IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems*.

3 Terms and definitions, notations, and conventions

3.1 Terms and definitions

3.1.1 Terms and definitions provided in Part 1

For the purposes of this part of ISO/IEC 13249, the terms and definitions given in ISO/IEC 13249-1 apply.

3.1.2 Terms and definitions provided in Part 3

For the purposes of this part of ISO/IEC 13249, the following terms and definitions apply.

3.1.2.1
0-dimensional geometry
a geometry with a geometric dimension of 0 (zero)

3.1.2.2
1-dimensional geometry
a geometry with a geometric dimension of 1 (one)

3.1.2.3
2-dimensional geometry
a geometry with a geometric dimension of 2

3.1.2.4
3-dimensional geometry
a geometry with a geometric dimension of 3

3.1.2.5
angle
a measure of the inclination of two surfaces, or of two coplanar lines, with respect to each other

Note 1 to entry: The angle may be measured by the arc of a circle intercepted between the two lines forming the angle, the center of the circle being the point of intersection). The value of an angle can be expressed in degrees, in degrees, minutes, and seconds, in radians, or in gradians.

3.1.2.6
azimuth
the angle, measured clockwise, between a given direction and a reference direction

Note 1 to entry: The reference direction is usually true north (North Azimuth) or true south (South Azimuth), and the angle is positive and less than 360 degrees or the equivalent in other angle measures.

3.1.2.7
bearing
a representation of a geographic heading that is given by a rotation measured from True North or True South towards East or West

Note 1 to entry: A bearing is specified with three parts: the prefix is either 'N' or 'S' for North or South; an angle in the range of 0 (zero) to 90 degrees, 0 to $\pi/2$ radians, or 0 to 100 gradians; and a suffix of 'E' or 'W' for East or West. For example, a direction of Northeast is defined as the bearing N 45 E, where 45 is in degrees. A bearing of S 30 E is 30 degrees measured counterclockwise from due South and is equivalent to a North azimuth of 150 degrees.

3.1.2.8
boundary of a curve
if the curve is closed, then the empty set; otherwise the set consisting of the end points of the curve

3.1.2.9
boundary of a point
the empty set

3.1.2.10
boundary of a solid
the set of surfaces that delineate the edge of the solid, including interior and exterior shells

3.1.2.11
boundary of a surface
the set of curves that delineate the edge of the surface, including interior and exterior rings

3.1.2.12
break void
void whose b*P03*-oundary elevations take precedence over existing surface elevations

3.1.2.13
breakline
discontinuity in the slope of a TIN surface

Note 1 to entry: When inserting a breakline into a surface, the elevations along the breakline take precedence over the original elevations on the surface.

3.1.2.14
closed curve
a curve such that its start point is equal to its end point

3.1.2.15
closed surface
a surface that is isomorphic to the surface of a sphere, or some torus

3.1.2.16
closure
a topological function to cause an open point-set to include its boundary making the point-set topologically closed

3.1.2.17
control contour
breakline having a constant elevation throughout its length

3.1.2.18
control point
position used in the construction of a geometry that partially controls its shape but does not necessarily lie on the geometry

Note 1 to entry: A center of an arc is a control point; poles in b-spline curves are control points.

3.1.2.19
data point
position used in the construction of a geometry that lies on the geometry

Note 1 to entry: The vertices in a line string are data points, the points used to construct a polynomial spline are data points. Data points can be used as control points, but are often derived after the geometry is constructed.

3.1.2.20
degree
a unit of measurement for angles such that there are 360 degrees in a circle

3.1.2.21
degrees, minutes, and seconds representation
a system of measurement for angles in which fractions of a degree can be expressed in terms of minutes and seconds

EXAMPLE *180 00 00.0 for an angle of π radians*

Note 1 to entry: For angles less than zero, only the degrees part is negative, however the positive minutes and seconds values are interpreted as increasing the negative-ness of the angle. An angle of –180 30 00.0 shall be interpreted to mean –(180 30 00.0).

3.1.2.22
dimension
geometric dimension

3.1.2.23
direction
the heading from one place to another

Note 1 to entry: The heading is usually given as an azimuth, but may be given as a bearing

3.1.2.24
distance between two geometries
the minimum of the distances between all pairs of points composed of one point from each of the two geometries

let g_1 and g_2 be two geometries,
dist(g_1, g_2) = Min ({ distance between p and q | $p \in g_1 \wedge q \in g_2$ })

3.1.2.25
distance between two points
the minimum of the lengths of all curves connecting two points

3.1.2.26
drape void
void whose boundary elevations are ignored

Note 1 to entry: When inserting a drape void into a surface, the elevation of the surface takes precedence over any elevations which may have been included in the drape void boundary.

3.1.2.27
geometry
the shape and geographic location of a feature

3.1.2.28
GML
Geography Markup Language as defined in ISO 19136

3.1.2.29
GML representation
an XML element that is valid against an XML element definition in the Geometry Schemas as defined in ISO 19136

3.1.2.30
gradian
a unit of measurement for angles such that there are 400 gradians in a circle

3.1.2.31
heading
a geographic direction that is measured as a rotation from some reference direction

3.1.2.32
heal
topological operation which amalgamates two edges (or links) into one by deleting their common node or which amalgamates two faces into one by deleting their common edge

Note 1 to entry: Heal is the reverse operation of split. When healing two successive edges (or links), the connecting node is deleted and a single edge (or link) replaces the original two. When healing two adjacent faces, their common edge is deleted and a single face replaces the original two.

3.1.2.33
intersection
(of two geometries) the set of points that are common to both geometries

3.1.2.34
isolated edge
an edge which is not part of the boundary of any face and is not connected to any other edge

3.1.2.35
isolated link
a link which is not connected to any other link

3.1.2.36
linear ring
a linestring that is closed and simple

3.1.2.37
linestring
a curve with linear interpolation between control points

3.1.2.38
link
one-dimensional topological primitive that represents a relationship between two nodes

3.1.2.39
logical network
a network which contains connectivity information but no geometric information

3.1.2.40
minute
a unit of measurement for angles such that there are 60 minutes in a degree

3.1.2.41
mod 2 union rule
operation on a multiset producing a resultant set consisting only of those elements occurring an odd number of times in the original multiset

Note 1 to entry: For example, by applying the mod 2 union rule to the multiset: A = { 10, 20, 20, 30, 30, 30, 40, 40, 40, 40 }, the result is the set: R = { 10, 30 }. In this example, R contains element 10 and 30 because these elements occur an odd number of times in A. Element 20 and 40 are not in R because they occur an even number of times in A.

3.1.2.42
network
set of nodes and links

3.1.2.43
non-closed curve
a curve such that its start point is not equal to its end point

3.1.2.44
non-universal face
any face other than the universal face

3.1.2.45
North azimuth
an azimuthal heading whose rotation is measured clockwise from True North

3.1.2.46
patch
planar component of a polyhedral surface

3.1.2.47
planar graph
graph that can be drawn in a plane without any of its edges crossing

3.1.2.48
point set
the representation of a geometry as a finite set or infinite set of points

Note 1 to entry: Mathematical set intersection ($\cap$), set union ($\cup$) and set difference (—) operations work on point sets.

3.1.2.49
polygon
a surface that is defined using linear rings to define its boundary

3.1.2.50
radian
a unit of measurement for angles such that there are 2pi radians in a circle

3.1.2.51
random point
point of known elevation from which a TIN surface can be generated

3.1.2.52
(regular) void
area within a TIN surface which signifies that the triangles within (or originally within) this area are voided

Note 1 to entry: When inserting a (regular) void into a surface, the vertices of the void boundary are added to the other points in the surface and re-triangulation (perhaps only local to the boundary) occurs.

3.1.2.53
ring
a curve that is closed and simple

3.1.2.54
rotation
an angle with a specific sense, which may be either clockwise or counterclockwise

3.1.2.55
second
a unit of measurement for angles such that there are 60 seconds in a minute

3.1.2.56
slope
<line segment> ratio of (a) the absolute difference in z coordinate values of the two end points to (b) the distance between them, ignoring their z coordinates

3.1.2.57
slope
<triangle> maximal slope of any line segment in a triangle

3.1.2.58
soft break
surface discontinuity which behaves as a breakline except that contour lines may be smoothed where they cross a soft break

3.1.2.59
South azimuth
an azimuthal heading whose rotation is measured clockwise from True South

3.1.2.60
spatial network
a network which contains both connectivity and geometric information

3.1.2.61
spatially 2D equals
a 2D test for equivalence between geometry values

Note 1 to entry: Ignoring z and m coordinate values, if the point sets of two geometry values are equal, then the geometries are spatially 2D equal.

3.1.2.62
spatially 3D equals
a 3D test for equivalence between geometry values

Note 1 to entry: Considering their z (but not m) coordinate values, if the point sets of two geometry values are equal, then the geometries are spatially 3D equal.

3.1.2.63
split
topological operation which creates two edges (or links) from one by inserting a node or which creates two faces from one by inserting an edge.

Note 1 to entry: Split is the reverse operation of heal.

3.1.2.64
stop line
indicator of places where the contour lines of a TIN surface are irregular or not continuous

EXAMPLE *the slope from the top to the bottom of a cliff*

3.1.2.65
TIN
Triangular Irregular Network as defined in ISO 19107

3.1.2.66
topologically closed
a characteristic of a geometry type that every value includes its own boundary

3.1.2.67
topology-geometry
a model assuming full planar topology, comprised of nodes, edges, and faces

3.1.2.68
topology-network
a model for linear applications including node and link topological primitives

3.1.2.69
True North
a geographic reference direction towards the Earth's geographic North pole

3.1.2.70
True South
a geographic reference direction towards the Earth's geographic South pole

3.1.2.71
unit of measure
defined quantity in which dimensioned parameters are expressed

3.1.2.72
universal face
a face containing everything else in a topology-geometry that is exterior to all other faces in that topology-geometry

3.1.2.73
voided area
within a TIN surface, a collection of contiguous triangles that remain in the surface but are designated as being void

Note 1 to entry: a voided area is bounded by a break void, drape void, or void.

3.1.2.74
XML element
an element as defined by ISO 19136

3.1.3 Terms and definitions taken from ISO 19107

For the purposes of this part of ISO/IEC 13249, the following terms defined in ISO 19107 apply.

a) boundary
b) buffer
c) computational topology
d) connected
e) connected node
f) convex hull of a geometric object
g) coordinate
h) coordinate dimension
i) coordinate reference system
j) coordinate system
k) curve
l) cycle
m) direct position
n) directed edge

o) directed face
p) directed topological object
q) edge
r) end node
s) end point
t) exterior
u) face
v) geometric complex
w) geometric dimension
x) geometric object
y) geometric primitive
z) homomorphism
aa) interior
ab) isolated node
ac) isomorphism
ad) node
ae) point
af) shell
ag) solid
ah) start node
ai) start point
aj) surface
ak) topological complex
al) topological object
am) topological primitive

3.1.4 Terms and definitions taken from ISO 19111

For the purposes of this part of ISO/IEC 13249, the following terms defined in ISO 19111 apply.

a) datum
b) ellipsoid
c) flattening
d) geodetic coordinate system, ellipsoidal coordinate system
e) height
f) meridian
g) prime meridian, zero meridian
h) projected coordinate system
i) semi-major axis
j) semi-minor axis

3.1.5 Terms and definitions taken from ISO 19148

For the purposes of this document, the following terms defined in ISO 19148 apply.

a) attribute event
b) attributed feature
c) feature
d) feature event
e) linear element
f) linear referencing
g) Linear Referencing Method
h) Linear Referencing System
i) linear segment
j) linearly located
k) linearly located event
l) linearly referenced location
m) located feature
n) locating feature
o) spatial position

3.2 Notations

3.2.1 Notations provided in Part 1

For the purposes of this part of ISO/IEC 13249, the notations given in ISO/IEC 13249-1 apply.

3.2.2 Notations provided in Part 3

This part of ISO/IEC 13249 uses the prefix 'ST_' for user-defined type, attribute, SQL-invoked routine table and view names.

This part of ISO/IEC 13249 uses the prefix 'ST_Private' for names of certain attributes. The use of 'ST_Private' indicates that the attribute is not for public use.

The real number mathematical constant that represents the circumference of a circle with unit diameter is notated as “π”. This number is transcendental and cannot be represented exactly in any algebraic form, so the precision is implementation-defined.

This part of ISO/IEC 13249 uses the symbols in Table 1 — Symbols.

Table 1 — Symbols

Symbols	Meaning
$\emptyset$	empty set
$\cap$	Intersection
$\cup$	union
—	difference
$\in$	is a member of
$\notin$	is not a member of
$\subset$	is a proper subset of
$\subseteq$	is a subset of
$\Leftrightarrow$	if and only if
$\Rightarrow$	implies
$\forall$	for all
{ *x* | ... }	set of all *x* such that ...
$\wedge$	and
$\vee$	or
$\neg$	not

3.3 Conventions

For the purposes of this part of ISO/IEC 13249, the conventions given in ISO/IEC 13249-1 apply.

3.4 Extended BNF notation for WKT and WKB

For defining Well-Known Text and Well-Known Binary strings, the conventions given in ISO/IEC 9075-1, Subclause 6.2 “Notation provided in this International Standard” shall apply. The following Clauses in this standard define well-known text:

geometry: 5.1.67 “<well-known text representation>”

spatial reference system: 14.1.9 “<spatial reference system>”

linear referencing: 15.14 “Linear Referencing Well-Known Text”

angle: 16.1.21 “<angle text representation>”

direction: 16.2.23 “<direction text representation>”

vector: 17.2.22 “<well-known text representation>”

4 Concepts

4.1 Concepts provided in Part 1

ISO/IEC 13249-1 Information Technology - Database Languages — SQL Multimedia and Application Packages — Part 1: Framework, clause 4 “Concepts” contains concepts that are generally applicable to multiple parts of ISO/IEC 13249, including this Part 3: Spatial. Additional concepts specific to Part 3: Spatial are included in the subsequent subclauses of clause 4 “Concepts” of this part.

4.2 Geometry Types

The following geometry types are defined: ST_Geometry, ST_Point, ST_Curve, ST_LineString, ST_CircularString, ST_Circle, ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve, ST_CompoundCurve, ST_Surface, ST_CurvePolygon, ST_Polygon, ST_Triangle, ST_PolyhdrlSurface, ST_TIN, ST_CompoundSurface, ST_Solid, ST_BRepSolid, ST_GeomCollection, ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface, and ST_MultiPolygon. ST_Geometry and its subtype family constitute the *geometry type hierarchy*, which is visually described in Annex E, "Geometry Type Hierarchy".

ST_Geometry, ST_Curve, ST_Surface, and ST_Solid are not instantiable types. No constructor functions are defined for these types.

ST_Point, ST_LineString, ST_CircularString, ST_Circle, ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve, ST_CompoundCurve, ST_CurvePolygon, ST_Polygon, ST_Triangle, ST_PolyhdrlSurface, ST_TIN, ST_CompoundSurface, ST_BRepSolid, ST_GeomCollection, ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface, and ST_MultiPolygon are instantiable and have constructor functions.

Any geometry type can be used as the type for a column. Declaring a column to be of a particular type implies that any value of the type or of any of its subtypes can be stored in the column.

ST_TINElement is a support type. It is used by the ST_TIN geometry type to specify information about the TIN surface, other than just its triangles. Each ST_TINElement contains an ST_Geometry.

ST_AffinePlacement and ST_Knot are also support types, used to specify certain ST_Curve types. They are not themselves subtypes of ST_Geometry.

4.2.1 ST_Geometry

The ST_Geometry type is the maximal supertype of the geometry type hierarchy. The ST_Geometry type is not instantiable. The instantiable subtypes of the ST_Geometry type are 0-dimensional geometry, 1-dimensional geometry, 2-dimensional geometry, and 3-dimensional geometry types that exist in two-dimensional coordinate space (R^2), three-dimensional coordinate space (R^3) or four-dimensional coordinate space (R^4). ST_Geometry values in R^2 have points with x and y coordinate values. ST_Geometry values in R^3 have points exclusively with x y and z coordinate values or exclusively with x, y and m coordinate values. ST_Geometry values in R^4 have points with x, y, z and m coordinate values.

The z coordinate is a coordinate of a point typically, but not necessarily, considered to represent altitude. The m coordinate is a coordinate of a point representing arbitrary measurement. ST_Geometry values that have the m coordinate value are key to supporting linear networking applications such as street routing, transportation, pipeline, telecommunications network, and utility management.

All instantiable types are defined so that all values are topologically closed (all ST_Geometry values include their boundary).

All locations in a geometry value are in the same spatial reference system.

In all routines, the geometric calculations are done in the spatial reference system of the first ST_Geometry value in the parameter list. If a routine returns an ST_Geometry value, then that value is in the spatial reference system of the first ST_Geometry value in the parameter list. Similarly, if the routine returns a measurement value such as length or area, then those values are returned in the spatial reference system of the first ST_Geometry value in the parameter list.

An implementation may define additional subtypes in the hierarchy that are outside the scope of this part of ISO/IEC 13249. An implementation shall preserve the subtype relationships between geometry types. Given two types *A* and *B* where *B* is an immediate subtype of *A*, an implementation may introduce another type *T* between types *A* and *B*, as an immediate supertype of *A*, or as an immediate subtype of *B.*

4.2.1.1 Methods on ST_Geometry

1) ST_Dimension: returns the dimension of an ST_Geometry value. The dimension of an ST_Geometry value is less than or equal to the coordinate dimension.
2) ST_CoordDim: returns the coordinate dimension of an ST_Geometry value. The coordinate dimension shall be the same as the coordinate dimension of the spatial reference system for the ST_Geometry value.
3) ST_GeometryType: returns the type of the ST_Geometry value as a CHARACTER VARYING value.
4) ST_SRID: observes and mutates the spatial reference system identifier of an ST_Geometry value.
5) ST_Transform: returns the ST_Geometry value in the specified spatial reference system, considering z and m coordinate values in the calculations and including them in the resultant geometry.
6) ST_IsEmpty: tests if an ST_Geometry value corresponds to the empty set.
7) ST_IsSimple: tests if an ST_Geometry value has no anomalous geometric points, such as self intersection or self tangency, ignoring z coordinate values in the determination. Subtypes of ST_Geometry will define the specific conditions that cause a value to be classified as simple.
8) ST_3DIsSimple: tests if an ST_Geometry value has no anomalous geometric points, such as self intersection or self tangency, considering z coordinate values in the determination. Subtypes of ST_Geometry will define the specific conditions that cause a value to be classified as simple.
9) ST_IsValid: tests if an ST_Geometry value is well formed.
10) ST_Is3D: tests whether an ST_Geometry value has z coordinates.
11) ST_IsMeasured: tests whether an ST_Geometry value has m coordinate values.
12) ST_LocateAlong: returns a derived geometry value that matches the specified m coordinate value, ignoring z coordinate values in the calculations and not including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
13) ST_3DLocateAlong: returns a derived geometry value that matches the specified m coordinate value, considering z coordinate values in the calculations and including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
14) ST_LocateBetween: returns a derived geometry value that matches the specified range of m coordinate values inclusively, ignoring z coordinate values in the calculations and not including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
15) ST_3DLocateBetween: returns a derived geometry value that matches the specified range of m coordinate values inclusively, considering z coordinate values in the calculations and including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
16) ST_Boundary: returns the boundary of an ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
17) ST_3DBoundary: returns the boundary of an ST_Geometry value, considering z coordinate values in the calculations and including them in the resultant geometry.
18) ST_Envelope: returns the bounding rectangular polygon of an ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
19) ST_EnvelopeAsPts: returns the minimum and maximum coordinate values of an ST_Geometry value as two ST_Point values.
20) ST_MinX: returns the minimum x coordinate value.
21) ST_MaxX: returns the maximum x coordinate value.
22) ST_MinY: returns the minimum y coordinate value.

23) ST_MaxY: returns the maximum y coordinate value.

24) ST_MinZ: returns the minimum z coordinate value.

25) ST_MaxZ: returns the maximum z coordinate value.

26) ST_MinM: returns the minimum m coordinate value.

27) ST_MaxM: returns the maximum m coordinate value.

28) ST_ConvexHull: returns the convex hull of an ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

29) ST_Buffer: returns the ST_Geometry value that represents all points whose distance from any point of an ST_Geometry value is less than or equal to a specified distance, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

30) ST_Intersection: returns the ST_Geometry value that represents the point set intersection of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

    Given two geometries *a* and *b*, ST_Intersection is defined as:

    a.ST_Intersection(*b*) ⇔ Closure($a \cap b$)

31) ST_3DIntersection: returns the ST_Geometry value that represents the point set intersection of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

32) ST_Union: returns the ST_Geometry value that represents the point set union of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

    Given two geometries *a* and *b*, ST_Union is defined as:

    a.ST_Union(*b*) ⇔ Closure($a \cup b$)

33) ST_3DUnion: returns the ST_Geometry value that represents the point set union of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

34) ST_Difference: returns the ST_Geometry value that represents the point set difference of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

    Given two geometries *a* and *b*, ST_Difference is defined as:

    a.ST_Difference(*b*) ⇔ Closure(*a* — *b*)

35) ST_3DDifference: returns the ST_Geometry value that represents the point set difference of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

36) ST_SymDifference: returns the ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

    Given two geometries *a* and *b*, ST_SymDifference is defined as:

    a.ST_SymDifference(*b*) ⇔ Closure(*a* — *b*) ∪ Closure(*b* — *a*) ⇔
    a.ST_Difference(*b*).ST_Union(*b*.ST_Difference(*a*))

37) ST_3DSymDifference: returns the ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

38) ST_Distance: returns the distance between two geometries, ignoring z and m coordinate values in the calculations.

39) ST_3DDistance: returns the distance between two geometries, considering z coordinate values in the calculations.

40) ST_WKTToSQL: returns the ST_Geometry value for the specified well-known text representation.

41) ST_AsText: returns the well-known text representation for the specified ST_Geometry value.

42) ST_WKBToSQL: returns the ST_Geometry value for the specified well-known binary representation.

43) ST_AsBinary: returns the well-known binary representation for the specified ST_Geometry value.

44) ST_GMLToSQL: returns the ST_Geometry value for the specified GML representation.

45) ST_AsGML: returns the GML representation for the specified ST_Geometry value.

4.2.1.2 Functions on ST_Geometry

1) ST_GeomFromText: returns an ST_Geometry value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Geometry.

2) ST_GeomFromWKB: returns an ST_Geometry value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Geometry.

3) ST_GeomFromGML: returns an ST_Geometry value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Geometry.

4.2.1.3 Ordering on ST_Geometry

1) ST_OrderingEquals: is the equals only ordering definition for the ST_Geometry type.

4.2.1.4 SQL Transforms on ST_Geometry

1) ST_WellKnownText: is the SQL Transform group that transforms an ST_Geometry value to and from a well-known text representation in a CHARACTER LARGE OBJECT value.

2) ST_WellKnownBinary: is the SQL Transform group that transforms an ST_Geometry value to and from a well-known binary representation in a BINARY LARGE OBJECT value.

3) ST_GML: is the SQL Transform group that transforms an ST_Geometry value to and from a GML representation in a CHARACTER LARGE OBJECT value.

4.2.1.5 Casts on ST_Geometry

The ST_Geometry type supports data conversion of a value from a source value (SV) of data type (SDT) to a target data type (TDT). SDT and TDT are instantiable subtypes of ST_Geometry. The supported types are described in Table 4 — Supported Casts. The SDT and TDT codes are data type codes (DT) defined in Table 2 — Data Type Codes. The cast codes used in Table 4 — Supported Casts are defined in Table 3 — Cast Codes.

NOTE For CX, casts are only supported in kind; casts from one CX Data Type to another CX Data Type are not supported.

Table 2 — Data Type Codes

DT	Data Type
P	ST_Point
C	ST_Curve
LS	ST_LineString
CX	any ST_Curve other than ST_LineString or ST_CompoundCurve
CC	ST_CompoundCurve
S	ST_Surface
CP	ST_CurvePolygon
PY	ST_Polygon
T	ST_Triangle
PS	ST_PolyhdrlSurface
TN	ST_TIN
CS	ST_CompoundSurface
BS	ST_BRepSolid
GC	ST_GeomCollection
MP	ST_MultiPoint
MC	ST_MultiCurve
MLS	ST_MultiLineString
MS	ST_MultiSurface
MPY	ST_MultiPolygon

Table 3 — Cast Codes

Cast Code	Description
Y	The cast from the source type to the target type is supported
E	If the value is an empty set, then the cast from the source type to the target type is supported.
1-*DT*	If the value is an empty set or the value contains 1 element of type *DT*, then the cast from the source type to the target type is supported. The *DT* codes are defined in Table 2 — Data Type Codes.
1-*DT1* 1-*DT2*	If the value is an empty set or the value contains 1 element of type *DT1* which has 1 element of type *DT2*, then the cast from the source type to the target type is supported. The *DT1* and *DT2* codes are defined DT codes in Table 2 — Data Type Codes.
1-*DT1* 4-*DT2*	If the value is an empty set or the value contains 1 element of type *DT1* which has 4 elements of type *DT2*, then the cast from the source type to the target type is supported. The *DT1* and *DT2* codes are defined DT codes in Table 2 — Data Type Codes.
n-*DT*	If the value is an empty set or the value contains 1 or more elements of type *DT*, then the cast from the source type to the target type is supported. The *DT* codes are defined in Table 2 — Data Type Codes.

Table 4 — Supported Casts

SDT	TDT																
	P	LS	CX	CC	CP	PY	T	PS	TN	CS	BS	GC	MP	MC	MLS	MS	MP Y
P	Y	E	E	E	E	E	E	E	E	E	E	Y	Y	E	E	E	E
LS	E	Y	E	Y	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
CX	E	Y	Y	Y	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
CC	E	Y	1-CX	Y	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
CP	E	E	E	E	Y	Y	1-LS 4-P	Y	1-LS 4-P	Y	E	Y	E	E	E	Y	Y
PY	E	E	E	E	Y	Y	1-LS 4-P	Y	1-LS 4-P	Y	E	Y	E	E	E	Y	Y
T	E	E	E	E	Y	Y	Y	Y	Y	E	E	Y	E	E	E	Y	Y
PS	E	E	E	E	1-PY	1-PY	1-T	Y	E	Y	E	Y	E	E	E	Y	1-PY
TN	E	E	E	E	1-T	1-T	1-T	Y	Y	E	E	Y	E	E	E	Y	1-T
CS	E	E	E	E	1-PS 1-CP	1-PS 1-CP	1-PS 1-T	1-PS	E	Y	E	Y	E	E	E	Y	1-PS 1-PY
BS	E	E	E	E	E	E	E	E	E	E	Y	Y	E	E	E	E	E
GC	1-P	1-C	1-CX	1-C	1-CP	1-CP	1-T	1-PS	1-TN	1-CS	1-BS	Y	n-P	n-C	n-C	n-S	n-S
MP	1-P	E	E	E	E	E	E	E	E	E	E	Y	Y	E	E	E	E
MC	E	1-C	1-CX	1-C	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
MLS	E	1-C	E	1-C	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
MS	E	E	E	E	1-CP	1-CP	1-T	1-PS	1-TN	1-S	E	Y	E	E	E	Y	Y
MPY	E	E	E	E	1-PY	1-PY	1-T	1-PY	E	1-PY	E	Y	E	E	E	Y	Y

4.2.1.6 Use of Z and M coordinate values

An ST_Point value may include a z coordinate value. The z coordinate value traditionally represents the third dimension (i.e. 3D). In a Geographic Information System (GIS) this may be height above or below sea level. For example: A map might have a point identifying the position of a mountain peak by its location on the earth, with the x and y coordinate values, and the height of the mountain, with the z coordinate value.

An ST_Point value may include an m coordinate value. The m coordinate value allows the application environment to associate some measure with the point values. For example: A stream network may be modeled as a multilinestring value with the m coordinate values measuring the distance from the mouth of the stream. ST_LocateBetween may be used to find all the parts of the stream that are between 10 and 12 kilometers from the mouth. There are no constraints on the m coordinate values in an ST_Geometry (e.g. the m coordinate values do not have to be continually increasing along an ST_LineString value).

As a general rule, unless explicity specified otherwise, when z or m coordinate values are present:

1) Observer methods that return ST_Point values shall include the z and m coordinate values.
2) Mutator methods shall maintain the ST_Point value z and m coordinate values.
3) Spatial operations shall not consider the z or m coordinate values in calculations (e.g. ST_Equals, ST_Length).
4) Routines which generate new geometry values (eg. ST_Intersection) shall not consider the z or m coordinate values.
5) Cast and transform routines shall maintain the z and m coordinate values.
6) Routines converting to or from WKT, WKB, and GML shall honor the z and m coordinate values when possible.
7) Routines whose name begins with "ST_3D" (e.g., ST_3DEquals, ST_3DIntersection) shall consider the z coordinate values and if returning a geometry value, shall include z coordinate values in the returned geometry.

When this general rule may lead to ambiguous behavior, the behavior is made explicit in the Description section of the routine specification.

4.2.1.7 Measures on ST_Geometry

The ST_LocateAlong and ST_LocateBetween methods derive geometry values from the given geometry that match a measure *M* or a specific range of measures from the start measure, *SM,* to the end measure, *EM*. The ST_LocateAlong method is a variation of the ST_LocateBetween method where the *SM* and *EM* are equal to *M*.

4.2.1.7.1 Empty Sets

A null value is returned when ST_LocateAlong or ST_LocateBetween are invoked on an empty set.

4.2.1.7.2 Geometry values without m coordinate values.

An empty set of type ST_Point is returned for geometry values without m coordinate values.

4.2.1.7.3 0-dimensional geometry values

Only points in the 0-dimensional geometry values with m coordinate values between *SM* and *EM* inclusively are returned as a multipoint value. If no matching m coordinate values are found, then an empty set of type ST_Point is returned.

For example:

a) If ST_LocateAlong is invoked with an *M* value of 4 on an ST_MultiPoint value with well-known text representation:

    multipoint m(1 0 4, 1 1 1, 1 2 2, 3 1 4, 5 3 4)

   then the result is the following ST_MultiPoint value with well-known text representation:

    multipoint m(1 0 4, 3 1 4, 5 3 4)

b) If ST_LocateBetween is invoked with an *SM* value of 2 and an *EM* value of 4 on an ST_MultiPoint value with well-known text representation:

multipoint m(1 0 4, 1 1 1, 1 2 2, 3 1 4, 5 3 5, 9 5 3, 7 6 7)

then the result is the following ST_MultiPoint value with well-known text representation:

multipoint m(1 0 4, 1 2 2, 3 1 4, 9 5 3)

c) If ST_LocateBetween is invoked with an *SM* value of 1 and an *EM* value of 4 on an ST_Point value with well-known text representation:

point m(7 6 7)

then the result is the following ST_Point value with well-known text representation:

point m empty

d) If ST_LocateBetween is invoked with an *SM* value of 7 and an *EM* value of 7 on an ST_Point value with well-known text representation:

point m(7 6 7)

then the result is the following ST_MultiPoint value with well-known text representation:

multipoint m(7 6 7)

4.2.1.7.4 1-dimensional geometry value

Interpolation is used to determine any points on the 1-dimensional geometry with an m coordinate value between SM and EM inclusively. The implementation-defined interpolation algorithm is used to estimate values between measured values, usually using a mathematical function. For example, given a measure of 6 and a 2-point linestring where the m coordinate value of the start point is 4 and the m coordinate value of the end point is 8, since 6 is halfway between 4 and 8, the interpolation algorithm would be a point on the linestring halfway between the start and end points. The interpolation is within a line segment and not across line segments in an ST_Curve. The interpolation is within an ST_Curve element and not across ST_Curve elements in an ST_MultiCurve. Interpolation shall be based on the length of the 1-dimensional geometry.

The results are produced in a geometry collection. If there are consecutive points in the 1-dimensional geometry with an m coordinate value between SM and EM inclusively, then a curve value element is added to the geometry collection to represent the curve elements between these consecutive points. Any disconnected points in the 1-dimensional geometry value with m coordinate values between SM and EM inclusively are also added to the geometry collection. If no matching m coordinate values are found, then an empty set of type ST_Point is returned.

For example:

a) If ST_LocateAlong is invoked with an M value of 4 on an ST_LineString value with well-known text representation:

linestring m(1 0 0, 3 1 4, 5 3 4, 5 5 1, 5 6 4, 7 8 4, 9 9 0)

then the result is the following ST_MultiLineString value with well-known text representation:

multilinestring m((3 1 4, 5 3 4), (5 6 4, 7 8 4))

b) If ST_LocateBetween is invoked with an SM value of 2 and an EM value of 4 on an ST_LineString value with well-known text representation:

linestring m(1 0 0, 1 1 1, 1 2 2, 3 1 3, 5 3 4, 9 5 5, 7 6 6)

then the result is the following ST_MultiLineString value with well-known text representation:

multilinestring m((1 2 2, 3 1 3, 5 3 4))

c) If ST_LocateBetween is invoked with an SM value of 6 and an EM value of 9 on an ST_LineString value with well-known text representation:

linestring m(1 0 0, 1 1 1, 1 2 2, 3 1 3, 5 3 4, 9 5 5, 7 6 6)

then the result is the following ST_MultiPoint value with well-known text representation:

multipoint m(7 6 6)

d) If ST_LocateBetween is invoked with an SM value of 1 and an EM value of 2 on an ST_LineString value with a well-known text representation:

linestring m(0 0 1, 2 2 3, 4 4 2)

then the result is the following ST_GeomCollection value with well-known text representation:

geometrycollection m(linestring m(0 0 1, 1 1 2), point m(4 4 2))

e) If ST_LocateBetween is invoked with an SM value of 2 and an EM value of 4 on an ST_MultiLineString value with well-known text representation:

multilinestring m((1 0 0, 1 1 1, 1 2 2, 3 1 3), (4 5 3, 5 3 4, 9 5 5, 7 6 6))

then the result is the following ST_MultiLineString value with well-known text representation:

multilinestring m((1 2 2, 3 1 3),(4 5 3, 5 3 4))

f) If ST_LocateBetween is invoked with an SM value of 1 and an EM value of 3 on an ST_LineString value with well-known text representation:

linestring m(0 0 0, 2 2 2, 4 4 4)

then the result is the following ST_MultiLineString value with well-known text representation:

multilinestring m((1 1 1, 2 2 2, 3 3 3))

g) If ST_LocateBetween is invoked with an SM value of 7 and an EM value of 9 on an ST_MultiLineString value with well-known text representation:

multilinestring m((1 0 0, 1 1 1, 1 2 2, 3 1 3), (4 5 3, 5 3 4, 9 5 5, 7 6 6))

then the result is the following ST_Point value with well-known text representation:

point m empty

4.2.1.7.5 2-dimensional geometry value

The computation for 2-dimensioinal geometries is implementation-defined.

4.2.1.7.6 3-dimensional geometry value

3-dimensioinal geometries do not have m coordinate values.

4.2.2 Spatial Relationships using ST_Geometry

The spatial relationships are methods that are used to test for the existence of a specified topological spatial relationship between two ST_Geometry values. The basic approach to comparing two ST_Geometry values is to make pair-wise tests of the intersections between the interiors, boundaries and exteriors of the two ST_Geometry values and to classify the relationship between the two ST_Geometry values based on the entries in the resulting intersection matrix.

The concepts of interior, boundary and exterior are well defined in general topology. These concepts can be applied in defining spatial relationships between 2-dimensional geometry values in *n*-dimensional coordinate space (R^n) where *n* is greater than 1 (one). In order to apply the concepts of interior, boundary and exterior to 0-dimensional geometry and 1-dimensional geometry values in R^n, a combinatorial topology approach is applied. This approach is based on the accepted definitions of the boundaries, interiors and exteriors for simplicial complexes and yields the following results.

The boundary of an ST_Geometry value is a set of ST_Geometry values of the next lower dimension. The boundary of an ST_Point value or an ST_MultiPoint value is the empty set. The boundary of a non-closed ST_Curve consists of the start and end ST_Point values; the boundary of a closed ST_Curve value is the empty set. The boundary of an ST_MultiCurve consists of those ST_Point values that are in the boundaries of an odd number of its element ST_Curve values. The boundary of an ST_Polygon value consists of its set of linear rings. The boundary of an ST_MultiPolygon value consists of the set of linear rings of its ST_Polygon values. The boundary of an arbitrary collection of geometries whose interiors are disjoint consists of geometries drawn from the boundaries of the element geometries by application of the mod 2 union rule.

The domain of ST_Geometry values considered consists of those values that are topologically closed. The interior of an ST_Geometry value consists of those points that are left when the boundary points are removed. The exterior of an ST_Geometry value consists of points not in the interior or boundary.

4.2.2.1 The Dimensionally Extended 9 Intersection Model

Given an ST_Geometry value *g*, let Interior(*g*), Boundary(*g*) and Exterior(*g*) represent the interior, boundary and exterior of *g,* respectively. The intersection of any two of Interior(*g*), Boundary(*g*) and Exterior(*g*) can result in a set of ST_Geometry values, *x*, of mixed dimension. For example, the intersection of the boundaries of two ST_Polygon values may consist of an ST_Point value and an ST_LineString value. Let *x*.ST_Dimension() return the maximum dimension (-1, 0, 1, or 2) of *x*, with a value of -1 corresponding to the dimension of the empty set (∅). A dimensionally extended nine intersection matrix (DE-9IM) is specified in Table 5 — DE-9IM.

Table 5 — DE-9IM

	Interior	Boundary	Exterior
Interior	(Interior(*a*) ∩ Interior(*b*)). ST_Dimension()	(Interior(*a*) ∩ Boundary(*b*)). ST_Dimension()	(Interior(*a*) ∩ Exterior(*b*)). ST_Dimension()
Boundary	(Boundary(*a*) ∩ Interior(*b*)). ST_Dimension()	(Boundary(*a*) ∩ Boundary(*b*)). ST_Dimension()	(Boundary(*a*) ∩ Exterior(*b*)). ST_Dimension()
Exterior	(Exterior(*a*) ∩ Interior(*b*)). ST_Dimension()	(Exterior(*a*) ∩ Boundary(*b*)). ST_Dimension()	(Exterior(*a*) ∩ Exterior(*b*)). ST_Dimension()

For topologically closed input geometries computing the dimension of the intersection of the interior, boundary and exterior sets does not have as a prerequisite the explicit computation and representation of these sets. For example, to compute if the interiors of two ST_Polygon values intersect and to ascertain the dimension of this intersection, it is not necessary to explicitly represent the interior of the two polygons (which are topologically open sets) as separate ST_Geometry values.

In most cases the dimension of the intersection value at a cell is highly constrained given the type of the two ST_Geometry values. For example, in the case of comparing an ST_LineString value with an ST_Polygon: the only possible values for the Interior-Interior cell are drawn from { -1, 1 }. In the case of comparing an ST_Polygon with an ST_Polygon, the only possible values for the Interior-Interior cell are drawn from { -1, 2 }. In such cases, no work beyond detecting the intersection is required.

A spatial relationship predicate can be formulated on ST_Geometry values that takes as input a pattern matrix representing the set of acceptable values for the DE-9IM for the two geometries. If the spatial relationship between the two ST_Geometry values corresponds to one of the acceptable values as represented by the pattern matrix, then the predicate returns 1 (one). Otherwise, 0 (zero).

The pattern matrix consists of a set of 9 pattern-values, one for each cell in the matrix. Let *p* be a pattern value. The possible values of *p* are { T, F, 0, 1, 2, * } and their meanings for any cell where *x* is the intersection set for the cell are:

Case:

a) if *p* = T, then *x*.ST_Dimension() ∈ { 0, 1, 2 }, i.e. *x* ≠∅

b) if *p* = F, then *x*.ST_Dimension() = -1, i.e. *x* = ∅

c) if *p* = 0, then *x*.ST_Dimension() = 0 (zero)

d) if *p* = 1, then *x*.ST_Dimension() = 1 (one)

e) if *p* = 2, then *x*.ST_Dimension() = 2

f) if *p* = *, then *x*.ST_Dimension() ∈ { -1, 0, 1, 2 }, i.e. any value

The pattern matrix can be represented as a <character string literal> with cardinality 9 representing the DE-9IM in row major order. See Subclause 5.3, "<literal>" in Part 2 of ISO/IEC 9075 for the definition of <character string literal>.

4.2.2.2 Common Spatial Relationships based on the DE-9IM

The ST_Relate method based on the pattern matrix enables a large number of spatial relationships to be tested and fine-tuned to the particular relationship being tested. However it is a low level building block and does not have a corresponding natural language equivalent. For this reason, commonly used spatial relationships have been specified: ST_Disjoint, ST_Intersects, ST_Touches, ST_Crosses, ST_Within, ST_Contains and ST_Overlaps. The theoretical basis for these relationships is limited to geometries having two coordinate dimensions. The z and m coordinate values will therefore be ignored when determining if these relationships hold between two geometry values. These spatial relationships have the following properties:

1) They are mutually exclusive.
2) They provide a complete covering of all topological cases.
3) They apply to spatial relationships between two geometries of either the same or different dimension.
4) Each predicate can be expressed in terms of a corresponding set of DE-9IM patterns.
5) Any realizable DE-9IM can be expressed as a Boolean expression over the 7 predicates, given the ST_Boundary method on the ST_Geometry type and the ST_StartPoint() and ST_EndPoint() method on the ST_Curve type.

4.2.2.3 Spatial Methods using ST_Geometry

1) ST_Equals: tests if an ST_Geometry value is spatially 2D equal to another ST_Geometry value. This test is for equivalence between two, possibly quite different, representations. The test may be limited by the resolution of the spatial reference system or the accuracy of the data. An implementation-defined tolerance may be provided such that two points are considered equal if the distance between the points is less that the tolerance.

   Given two geometries *a* and *b*, ST_Equals is defined as:

   a.ST_Equals(*b*) ⇔
   $(a — b) \cup (b — a) = \emptyset$ ⇔
   a.ST_SymDifference(*b*).ST_IsEmpty()

   When determining if two geometries are equal, z and m coordinate values are ignored in the calculations.

2) ST_3DEquals: tests if an ST_Geometry value is spatially 3D equal to another ST_Geometry value. When determining if two geometries are 3D equal, z (but not m) coordinate values are considered in the calculations.

3) ST_Relate: tests if an ST_Geometry value is spatially related to another ST_Geometry value by testing for intersections between the interior, boundary and exterior of the two ST_Geometry values as specified by the intersection matrix.

   When determining if two geometries are spatially related, z and m coordinate values are ignored in the calculations.

4) ST_Disjoint: tests if an ST_Geometry value is spatially disjoint from another ST_Geometry value.

   Given two geometries *a* and *b*, ST_Disjoint is defined as:

   a.ST_Disjoint(*b*) ⇔
   $a \cap b = \emptyset$

   Expressed in terms of the DE-9IM:

   a.ST_Disjoint(*b*) ⇔
   (Interior(*a*) ∩ Interior(*b*) = ∅) ∧
   (Interior(*a*) ∩ Boundary(*b*) = ∅) ∧
   (Boundary(*a*) ∩ Interior(*b*) = ∅) ∧
   (Boundary(*a*) ∩ Boundary(*b*) = ∅) ⇔
   a.ST_Relate(*b*, 'FF*FF****')

   When determining if two geometries are disjoint, z and m coordinate values are ignored in the calculations.

5) ST_3DDisjoint: tests if an ST_Geometry value is spatially 3D disjoint from another ST_Geometry value. When determining if two geometries are 3D disjoint, z (but not m) coordinate values are considered in the calculations.

6) ST_Intersects: tests if an ST_Geometry value spatially intersects another ST_Geometry value:

   Given two geometries *a* and *b*, ST_Intersects is defined as:

   a.ST_Intersects(*b*) ⇔
   Case *a*.ST_Disjoint(*b*) when 0 then 1 when 1 then 0 else NULL end

   When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

7) ST_3DIntersects: tests if an ST_Geometry value spatially 3D intersects another ST_Geometry value. When determining if two geometries 3D intersect, z (but not m) coordinate values are considered in the calculations.

8) ST_Touches: tests if an ST_Geometry value spatially touches another ST_Geometry value.

   Given two geometries *a* and *b*, ST_Touches is defined as:

   Case:

   a) If *a*.ST_Dimension() = 0 (zero) and *b*.ST_Dimension() = 0 (zero), then the null value

   b) Otherwise, *a*.ST_Touches(*b*) ⇔ (Interior(*a*) ∩ Interior(*b*) = ∅) ∧ (*a* ∩ *b*) ≠ ∅

   Expressed in terms of the DE-9IM:

   Case:

   a) If *a*.ST_Dimension() = 0 (zero) and *b*.ST_Dimension() = 0 (zero), then the null value

   b) Otherwise:

      a.ST_Touches(*b*) ⇔
      (Interior(*a*) ∩ Interior(*b*) = ∅) ∧
         ((Boundary(*a*) ∩ Interior(*b*) ≠ ∅) ∨
         (Interior(*a*) ∩ Boundary(*b*) ≠ ∅) ∨
         (Boundary(*a*) ∩ Boundary(*b*) ≠ ∅)) ⇔
      Case *a*.ST_Relate(*b*, 'FT*******') = 1 OR
         a.ST_Relate(*b*, 'F**T*****') = 1 OR
         a.ST_Relate(*b*, 'F***T****') = 1 when TRUE then 1 when FALSE then 0 else NULL end

   When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

9) ST_Crosses: tests if an ST_Geometry value spatially crosses another ST_Geometry value.

   Given two geometries *a* and *b*, ST_Crosses is defined as:

   Case:

   a) If *a*.ST_Dimension() = 2 or *b*.ST_Dimension() = 0 (zero), then the null value

   b) Otherwise:

      a.ST_Crosses(*b*) ⇔
      ((Interior(*a*) ∩ Interior(*b*)).ST_Dimension() <
         max(Interior(*a*).ST_Dimension(), Interior(*b*).ST_Dimension())) ∧ (*a* ∩ *b* ≠ *a*) ∧ (*a* ∩ *b* ≠ *b*)

   Expressed in terms of the DE-9IM:

   Case:

   a) If *a*.ST_Dimension() = 2 or *b*.ST_Dimension() = 0 (zero), then the null value

   b) If (*a*.ST_Dimension() = 0 (zero) and *b*.ST_Dimension() = 1 (one)) or
         (*a*.ST_Dimension() = 0 (zero) and *b*.ST_Dimension() = 2) or
         (*a*.ST_Dimension() = 1 (one) and *b*.ST_Dimension() = 2), then:

a.ST_Crosses(*b*) ⇔
(Interior(*a*) ∩ Interior(*b*) ≠ ∅) ∧ (Interior(*a*) ∩ Exterior(*b*) ≠ ∅) ⇔
a.ST_Relate(*b*, 'T*T******')

c) If *a*.ST_Dimension() = 1 (one) and *b*.ST_Dimension() = 1 (one), then:

a.ST_Crosses(*b*) ⇔
(Interior(*a*) ∩ Interior(*b*)).ST_Dimension() = 0 (zero) ⇔
a.ST_Relate(*b*, '0********')

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

10) ST_Within: tests if an ST_Geometry value is spatially within another ST_Geometry value.

Given two geometries *a* and *b*, ST_Within is defined as:

a.ST_Within(*b*) ⇔ (*a* ∩ *b* = *a*) ∧ (Interior(*a*) ∩ Exterior(*b*) = ∅)

Expressed in terms of the DE-9IM:

a.ST_Within(*b*) ⇔
(Interior(*a*) ∩ Interior(*b*) ≠ ∅) ∧
(Interior(*a*) ∩ Exterior(*b*) = ∅) ∧ (Boundary(*a*) ∩ Exterior(*b*) = ∅) ⇔
a.ST_Relate(*b*, 'T*F**F***')

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

11) ST_Contains: tests if an ST_Geometry value spatially contains another ST_Geometry value.

Given two geometries *a* and *b*, ST_Contains is defined as:

a.ST_Contains(*b*) ⇔ *b*.ST_Within(*a*)

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

12) ST_Overlaps: tests if an ST_Geometry value spatially overlaps another ST_Geometry value.

Given two geometries *a* and *b*, ST_Overlaps is defined as:

Case:

a) If (*a*.ST_Dimension() = 0 (zero) and *b*.ST_Dimension() = 0 (zero)) or
(*a*.ST_Dimension() = 1 (one) and *b*.ST_Dimension() = 1 (one)) or
(*a*.ST_Dimension() = 2 and *b*.ST_Dimension() = 2), then:

a.ST_Overlaps(*b*) ⇔
(Interior(*a*).ST_Dimension() = Interior(*b*).ST_Dimension() =
(Interior(*a*) ∩ Interior(*b*)).ST_Dimension()) ∧ (*a* ∩ *b* ≠ *a*) ∧ (*a* ∩ *b* ≠ *b*)

b) Otherwise, the null value

Expressed in terms of the DE-9IM:

Case:

a) If (*a*.ST_Dimension() = 0 (zero) and *b*.ST_Dimension() = 0 (zero)) or
(*a*.ST_Dimension() = 2 and *b*.ST_Dimension() = 2), then:

a.ST_Overlaps(*b*) ⇔
(Interior(*a*) ∩ Interior(*b*) ≠ ∅) ∧
(Interior(*a*) ∩ Exterior(*b*) ≠ ∅) ∧
(Exterior(*a*) ∩ Interior(*b*) ≠ ∅) ⇔
a.ST_Relate(*b*, 'T*T***T**')

b) If *a*.ST_Dimension() = 1 (one) and *b*.ST_Dimension() = 1 (one), then:

a.ST_Overlaps(b) $\Leftrightarrow$
((Interior(a) $\cap$ Interior(b)).ST_Dimension() = 1 (one)) $\wedge$
(Interior(a) $\cap$ Exterior(b) $\neq \emptyset$) $\wedge$ (Exterior(a) $\cap$ Interior(b) $\neq \emptyset$) $\Leftrightarrow$
a.ST_Relate(b, '1*T***T**')

c) Otherwise, the null value

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

4.2.2.4 GML Support for ST_Geometry

An ST_Geometry value may be represented as an XML element in the GML representation. This support does not include support for an XML document using any or all of the XML schemas defined in GML. The provided routines may be used to create ST_Geometry values from specific XML elements from the Geometry Schemas and to generate XML elements from ST_Geometry values. The mapping is defined in Table 14 — Mapping between ST_Geometry values and GML representation.

4.2.3 ST_Point

The ST_Point type is a subtype of ST_Geometry. The ST_Point type is instantiable. An ST_Point value is a 0-dimensional geometry and represents a single location. An ST_Point has an x coordinate value, a y coordinate value, an optional z coordinate value, and an optional m coordinate value. The boundary of an ST_Point value is the empty set. ST_Point values are simple.

4.2.3.1 Methods on ST_Point

1) ST_Point: returns an ST_Point value constructed from either:
   a) the well-known text representation of an ST_Point value;
   b) the well-known binary representation of an ST_Point value;
   c) the GML representation of an ST_Point value;
   d) the specified coordinate values.
2) ST_X: observes and mutates the x coordinate value of an ST_Point value.
3) ST_Y: observes and mutates the y coordinate value of an ST_Point value.
4) ST_Z: observes and mutates the z coordinate value of an ST_Point value.
5) ST_M: observes and mutates the m coordinate value of an ST_Point value.
6) ST_ExplicitPoint: returns the coordinate values as a DOUBLE PRECISION ARRAY value.

4.2.3.2 Functions on ST_Point

1) ST_PointFromText: returns an ST_Point value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Point.
2) ST_PointFromWKB: returns an ST_Point value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Point.
3) ST_PointFromGML: returns an ST_Point value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Point value.

4.2.4 ST_Curve

The ST_Curve type is a subtype of ST_Geometry. The ST_ Curve type is not instantiable. An ST_Curve value is a 1-dimensional geometry usually stored as a sequence of points. The subtype of ST_Curve specifies the form of the interpolation between points.

Topologically, an ST_Curve value is a 1-dimensional geometry that is the homomorphic image of a real, closed interval. An ST_Curve value is not simple if any interior point has the same location as another interior point or a point in the boundary.

An ST_Curve value is closed if its start point is equal to its end point. The boundary of a closed ST_Curve value is the empty set. An ST_Curve value that is simple and closed is called a *ring*. The boundary of a non-closed ST_Curve value consists of its start point and its end point. An ST_Curve value is defined to be topologically closed.

4.2.4.1 Methods on ST_Curve

1) ST_Length: returns the length of an ST_Curve value, ignoring z and m coordinate values in the calculations.
2) ST_3DLength: returns the length of an ST_Curve value, considering z coordinate values in the calculations.
3) ST_StartPoint: returns the ST_Point value that is the start point of an ST_Curve value including existing z and m coordinate values in the resultant geometry.
4) ST_EndPoint: returns the ST_Point value that is the end point of an ST_Curve value including existing z and m coordinate values in the resultant geometry.
5) ST_IsClosed: tests if an ST_Curve value is closed, ignoring z and m coordinate values in the calculations.
6) ST_3DIsClosed: tests if an ST_Curve value is closed, considering z (but not m) coordinate values in the calculations.
7) ST_IsRing: tests if an ST_Curve value is a ring, ignoring z coordinate values in the calculations.
8) ST_3DIsRing: tests if an ST_Curve value is a ring, considering z coordinate values in the calculations.
9) ST_CurveToLine: returns the ST_LineString value approximating the ST_Curve value, considering z and m coordinate values in the calculations and including z and m coordinate values in the resultant geometry.
10) ST_DistanceToPoint: returns the distance from the start of the curve measured along the curve to a point on the curve, ignoring z and m coordinate values in the calculations.
11) ST_3DDistanceToPt: returns the distance from the start of the curve measured along the curve to a point on the curve, considering z (but not m) coordinate values in the calculations.
12) ST_PointAtDistance: returns the ST_Point value that is the specified distance from the start of the curve measured along the curve, ignoring z coordinate values in the calculations and including an interpolated m (but not z) coordinate in the return value.
13) ST_3DPtAtDistance: returns the ST_Point value that is the specified distance from the start of the curve measured along the curve, considering z coordinate values in the calculations and including a z and interpolated m coordinate in the return value.
14) ST_PerpPoints: returns the geometry representing the perpendicular projection of the given point onto the curve, ignoring z coordinate values in the calculations and including interpolated m (but not z) coordinates in the resultant geometry.

4.2.5 ST_LineString

The ST_LineString type is a subtype of ST_Curve. The ST_LineString type is instantiable. An ST_LineString value has linear interpolation between ST_Point values. Each consecutive pair of ST_Point values defines a line segment. A line is an ST_LineString value with exactly two points. A linear ring is an ST_LineString value that is both closed and simple.

4.2.5.1 Methods on ST_LineString

1) ST_LineString: returns an ST_LineString value constructed from either:
   a) the well-known text representation of an ST_LineString value;
   b) the well-known binary representation of an ST_LineString value;
   c) the GML representation of an ST_LineString value;
   d) the specified ST_Point values.
2) ST_Points: observes and mutates the ST_Point collection in the ST_LineString value.
3) ST_NumPoints: returns the cardinality of the ST_Point collection in the ST_LineString value.
4) ST_PointN: returns the specified element in the ST_Point collection in the ST_LineString value.

4.2.5.2 Functions on ST_LineString

1) ST_LineFromText: returns an ST_LineString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LineString.
2) ST_LineFromWKB: returns an ST_LineString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_LineString.
3) ST_LineFromGML: returns an ST_LineString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML LineString or LineStringSegment representation of an ST_LineString value.

4.2.6 ST_CircularString

The ST_CircularString type is a subtype of ST_Curve. The ST_CircularString type is instantiable. An ST_CircularString value has circular interpolation between ST_Point values. This subtype of ST_Curve consists of one or more circular arc segments connected end to end. The first three points define the first segment. The first point is the start point of the arc. The second point is any intermediate point on the arc other than the start or end point. The third point is the end point of the arc and shall be distinct from the first point. Subsequent segments are defined by their intermediate and end points only, as the start point is implicitly defined as the previous segment's end point.

Let *CHORD1* be the line connecting the start point of a circular arc segment and the intermediate point on the segment. Let *CHORD2* be the line connecting the intermediate point with the end point of this arc segment. The center of the circular arc segment is located at the intersection of the perpendicular bisectors of *CHORD1* and *CHORD2*.

Let distance *R* be the radius of the circular arc segment, equal to the distance from the center of the circular arc segment to the start, intermediate, or end points.

The circular arc segment is the locus of points a distance *R* from the center of the arc, beginning at the start point and ending at the end point of the circular arc segment.

If the start, intermediate, and end points of an arc segment are collinear, then the resultant arc segment degenerates to a straight line for which center and radius are not defined. In this case, the circular arc segment is the locus of points defined by the straight line connecting the start and end points.

An ST_CircularString value with exactly three points is a circular arc. A circular ring is an ST_CircularString value that is both closed and simple.

An ST_CircularString may be specified by using a bulge factor. Each arc (curve segment) is defined by start and end points, a bulge and a normal. Since the arcs can be connected, start points of all but the first arc are not required, as they are the end points of the previous arc.

A single arc ST_CircularString may also be specified by specifying a single control point at the center point of the arc plus the radius and the start and end angles. This representation can be used only in 2D.

4.2.6.1 Methods on ST_CircularString

1) ST_CircularString: returns an ST_CircularString value constructed from either:
   a) the well-known text representation of an ST_CircularString value;
   b) the well-known binary representation of an ST_CircularString value;
   c) the GML representation of an ST_CircularString value;
   d) the GML ArcByBulge representation of an ST_CircularString value;
   e) the GML ArcStringByBulge representation of an ST_CircularString value;
   f) the GML ArcByCenterPoint representation of an ST_CircularString value;
   g) the specified ST_Point values;
   h) the specified ST_Point control point, DOUBLE PRECISION bulge and ST_Vector bulge normal ARRAY values;
   i) the specified ST_Point control (center) point, DOUBLE PRECISION radius and ST_Angle start and end angle values.
2) ST_Points: observes and mutates the ST_Point collection in the ST_CircularString value.
3) ST_NumPoints: returns the cardinality of the ST_Point collection in the ST_CircularString value.
4) ST_PointN: returns the specified element in the ST_Point collection in the ST_CircularString value.
5) ST_MidPointRep: returns the array of points which identify an ST_CircularString value including start, mid, and end points for each curve segment.
6) ST_NumSegments: returns the INTEGER number of curve segments (arcs) for the ST_CircularString.
7) ST_SegmentN: returns the Nth curve segment for the ST_CircularString as an ST_CircularString having a single curve segment.
8) ST_Bulge: returns the DOUBLE PRECISION bulge value for an ST_CircularString having a single curve segment.
9) ST_BulgeNormal: returns the ST_Vector bulge normal value for an ST_CircularString having a single curve segment.
10) ST_Center: returns the ST_Point center point value for an ST_CircularString having a single curve segment.
11) ST_Radius: returns the DOUBLE PRECISION radius value for an ST_CircularString having a single curve segment.
12) ST_StartAngle: returns the ST_Angle start angle value for an ST_CircularString having a single curve segment.
13) ST_EndAngle: returns the ST_Angle end angle value for an ST_CircularString having a single curve segment.

4.2.6.2 Functions on ST_CircularString

1) ST_CircularFromTxt: returns an ST_CircularString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CircularString.
2) ST_CircularFromWKB: returns an ST_CircularString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CircularString.
3) ST_CircularFromGML: returns an ST_CircularString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Arc, ArcString, ArcByBulge, ArcStringByBulge or ArcByCenter representation of an ST_CircularString.

4.2.7 ST_Circle

The ST_Circle type is a subtype of ST_Curve. The ST_Circle type is instantiable. An ST_Circle value has circular interpolation between ST_Point values.

An ST_Circle is a single arc which is simple and closed. It consists of a sequence of three unique, non-collinear control points. The arc is simply extended past the third control point until the first control point is encountered.

4.2.7.1 Methods on ST_Circle

1) ST_Circle: returns an ST_Circle value constructed from either:
   a) the well-known text representation of an ST_Circle value;
   b) the well-known binary representation of an ST_Circle value;
   c) the GML Circle representation of an ST_Circle value;
   d) the GML CircleByCenterPoint representation of an ST_Circle value;
   e) the specified ST_Point values;
   f) the specified ST_Point center point, DOUBLE PRECISION radius and ST_Vector normal vector values;
2) ST_Points: observes and mutates the ST_Point collection in the ST_Circle value.
3) ST_PointN: returns the specified element in the ST_Point collection in the ST_Circle value.
4) ST_Radius: returns the DOUBLE PRECISION radius of the ST_Circle value.
5) ST_Center: returns the ST_Point value representing the center of the ST_Circle value.
6) ST_Normal: returns the ST_Vector value representing the normal vector of the ST_Circle value.

4.2.7.2 Functions on ST_Circle

1) ST_CircleFromTxt: returns an ST_Circle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Circle.
2) ST_CircleFromWKB: returns an ST_Circle value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Circle.
3) ST_CircleFromGML: returns an ST_Circle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Circle or CircleByCenterPoint representation of an ST_Circle value.

4.2.8 ST_GeodesicString

An ST_GeodesicString consists of a sequence of two or more control points joined by geodesic curve segments. The control points are a sequence of positions between which the ST_GeodesicString is interpolated using geodesics from the geoid or ellipsoid of the coordinate reference system being used.

The ST_GeodesicString type is a subtype of ST_Curve. The ST_GeodesicString type is instantiable. An ST_GeodesicString value has geodesic interpolation. It is represented by the following attributes:

control points – an ST_Point ARRAY.

4.2.8.1 Methods on ST_GeodesicString

1) ST_GeodesicString: returns an ST_GeodesicString value constructed from either:
   a) the well-known text representation of an ST_GeodesicString value;
   b) the well-known binary representation of an ST_GeodesicString value;
   c) the GML representation of an ST_GeodesicString value;
   d) the specified ST_Point values.
2) ST_Points: observes and mutates the ST_Point collection in the ST_GeodesicString value.
3) ST_NumPoints: returns the cardinality of the ST_Point collection in the ST_GeodesicString value.

4) ST_PointN: returns the specified element in the ST_Point collection in the ST_GeodesicString value.

4.2.8.2 Functions on ST_GeodesicString

1) ST_GeodesicFromTxt: returns an ST_GeodesicString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_GeodesicString.
2) ST_GeodesicFromWKB: returns an ST_GeodesicString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_GeodesicString.
3) ST_GeodesicFromGML: returns an ST_GeodesicString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Geodesic or GeodesicString representation of an ST_GeodesicString value.

4.2.9 ST_EllipticalCurve

The ST_EllipticalCurve type is a subtype of ST_Curve. The ST_EllipticalCurve type is instantiable. An ST_EllipticalCurve value represents a single curve segment having elliptical interpolation. It is represented by the following attributes:

reference location – an affine mapping of type ST_AffinePlacement that describes the local coordinate system with origin at the elliptical curve center.

uAxisLength – the DOUBLE PRECISION elliptical curve axis length along the u axis of its local 2D coordinate system. This is commonly called the "semimajor axis", but is not required to be the larger axis.

vAxisLength – the DOUBLE PRECISION elliptical curve axis length along the v axis of its local 2D coordinate system. This is commonly called the "semiminor axis, but it is not required to be the smaller axis.

start angle – constructive parameter angle of type ST_Angle

end angle – constructive parameter angle of type ST_Angle

start m – (optional) measure value at the start of the curve

end m – (optional) measure value at the end of the curve

4.2.9.1 Methods on ST_EllipticalCurve

1) ST_EllipticalCurve: returns an ST_EllipticalCurve value constructed from either:
   a) the well-known text representation of an ST_EllipticalCurve value;
   b) the well-known binary representation of an ST_EllipticalCurve value;
   c) the GML representation of an ST_EllipticalCurve value;

      NOTE There is no normative GML type as of GML version 3.2.1. One has formally been proposed.
   d) the specified reference location ST_AffinePlacement, DOUBLE PRECISION uAxisLength and vAxisLength values, and the start and end ST_Angle values;
   e) the specified reference location ST_AffinePlacement, DOUBLE PRECISION uAxisLength and vAxisLength values, the start and end ST_Angle values and the DOUBLE PRECISION start and end measure values.
2) ST_RefLocation: observes and mutates the ST_AffinePlacement reference location value in the ST_EllipticalCurve value.
3) ST_UAxisLength: observes and mutates the DOUBLE PRECISION u axis length value of an ST_EllipticalCurve value.
4) ST_VAxisLength: observes and mutates the DOUBLE PRECISION v axis length value of an ST_EllipticalCurve value.
5) ST_StartAngle: observes and mutates the ST_Angle start angle value of an ST_EllipticalCurve value.

6) ST_EndAngle: observes and mutates the ST_Angle end angle value of an ST_EllipticalCurve value.
7) ST_StartM: observes and mutates the measure value at the start of an ST_EllipticalCurve value.
8) ST_EndM: observes and mutates the measure value at the end of an ST_EllipticalCurve value.

4.2.9.2 Functions on ST_EllipticalCurve

1) ST_EllipticFromTxt: returns an ST_EllipticalCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_EllipticalCurve.
2) ST_EllipticFromWKB: returns an ST_EllipticalCurve value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_EllipticalCurve.
3) ST_EllipticFromGML: returns an ST_EllipticalCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Ellipse or EllipticalCurve representation of an ST_EllipticalCurve value

   NOTE Once one is defined there.

4.2.10 ST_NURBSCurve

The ST_NURBSCurve type is a subtype of ST_Curve having a single, continuous curve segment. The ST_NURBSCurve type is instantiable. An ST_NURBSCurve value has piecewise polynomial of (x,y,z) position as a function of an interval in the knots space. The ST_NURBSCurve data is a "Non-Uniform Rational BSpline" as described by ISO 19107.

An ST_NURBSCurve value is represented by the following attributes:

degree – the INTEGER degree of the polynomials in the spline

control points – an ST_NURBSPoint ARRAY which contains ccontrol points which have been adjusted in consideration of their respective weight values.

knots – ST_Knot ARRAY which contains knot values and their respective multiplicities

start m – (optional) measure value at the start of the curve

end m – (optional) measure value at the end of the curve

4.2.10.1 Methods on ST_NURBSCurve

1) ST_NURBSCurve: returns an ST_NURBSCurve value constructed from either:
   a) the well-known text representation of an ST_NURBSCurve value;
   b) the well-known binary representation of an ST_NURBSCurve value;
   c) the GML BSpline representation of an ST_NURBSCurve value;
   d) the specified INTEGER degree, ST_NURBSPoint ARRAY control points and ST_Knot ARRAY knots;
   e) the specified INTEGER degree, ST_NURBSPoint ARRAY control points, ST_Knot ARRAY knots and the DOUBLE PRECISION start and end measure values.
2) ST_Degree: returns the degree of the ST_NURBSCurve value.
3) ST_ControlPoints: observes and mutates the control point collection in the ST_NURBSCurve value.
4) ST_Knots: observes and mutates the knot collection in the ST_NURBSCurve value.
5) ST_StartM: observes and mutates the measure value at the start of an ST_NURBSCurve value.
6) ST_EndM: observes and mutates the measure value at the end of an ST_NURBSCurve value.

4.2.10.2 Functions on ST_NURBSCurve

1) ST_NURBSFromTxt: returns an ST_NURBSCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_NURBSCurve.
2) ST_NURBSFromWKB: returns an ST_NURBSCurve value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_NURBSCurve.
3) ST_NURBSFromGML: returns an ST_NURBSCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML BSpline representation of an ST_NURBSCurve value.

4.2.11 ST_Clothoid

The ST_Clothoid type is a subtype of ST_Curve. The ST_Clothoid type is instantiable. An ST_Clothoid value represents a single curve segment having clothoid interpolation. A clothoid, or Cornu's spiral, is a plane curve whose curvature is proportional to the distance along the curve from its (single) point of inflection. (Curvature at a particular point along the curve is the inverse of the radius of the osculating circle at that point on the curve.)

An ST_Clothoid value is represented by the following attributes:

reference location – an affine mapping of type ST_AffinePlacement that places the curve defined by the Fresnel Integrals into the coordinate reference system of this curve

scale factor – of type DOUBLE PRECISION

start distance – the DOUBLE PRECISION arc length distance from the inflection point that will be the start point for the curve segment. It is the lower limit "t" used in the Fresnel integral and is the value of the constructive parameter of the curve segment at its start point.

end distance – the DOUBLE PRECISION arc length distance from the inflection point that will be the end point for the curve segment. It is the upper limit "t" used in the Fresnel integral and is the constructive parameter of the curve segment at its end point.

start m – (optional) measure value at the start of the curve

end m – (optional) measure value at the end of the curve

4.2.11.1 Methods on ST_Clothoid

1) ST_Clothoid: returns an ST_Clothoid value constructed from either:
   a) the well-known text representation of an ST_Clothoid value;
   b) the well-known binary representation of an ST_Clothoid value;
   c) the GML representation of an ST_Clothoid value;
   d) the specified ST_AffinePlacement reference location and the DOUBLE PRECISION values for scale factor, start distance and end distance;
   e) the specified ST_AffinePlacement reference location and the DOUBLE PRECISION values for scale factor, start distance, end distance and start and end measure values.
2) ST_RefLocation: observes and mutates the ST_AffinePlacement reference location value in the ST_Clothoid value.
3) ST_ScaleFactor: observes and mutates the DOUBLE PRECISION scale factor value of an ST_Clothoid value.
4) ST_StartDistance: observes and mutates the DOUBLE PRECISION start distance value of an ST_Clothoid value.
5) ST_EndDistance: observes and mutates the DOUBLE PRECISION end distance value of an ST_Clothoid value.
6) ST_StartM: observes and mutates the measure value at the start of an ST_Clothoid value.
7) ST_EndM: observes and mutates the measure value at the end of an ST_Clothoid value.

4.2.11.2 Functions on ST_Clothoid

1) ST_ClothoidFromTxt: returns an ST_Clothoid value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Clothoid.
2) ST_ClothoidFromWKB: returns an ST_Clothoid value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Clothoid.
3) ST_ClothoidFromGML: returns an ST_Clothoid value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Clothoid value.

4.2.12 ST_SpiralCurve

The ST_SpiralCurve type is a subtype of ST_Curve. The ST_SpiralCurve type represents a single curve segment having spiral interpolation. The ST_SpiralCurve type is instantiable. It is typically used to define curves used for railroad alignments. It is represented by the following attributes:

reference location – an affine mapping of type ST_AffinePlacement that places the spiral into the coordinate reference system of this curve. The spiral start point is the origin in the placement coordinates, and the initial direction is along the positive x axis.

length – the DOUBLE PRECISION length of the curve

start curvature – the DOUBLE PRECISION start curvature value

end curvature – the DOUBLE PRECISION end curvature value

spiral type – the type of spiral, initially limited to clothoid, bloss, biquadratic, sine and cosine as a CHARACTER VARYING value.

start m – (optional) measure value at the start of the curve

end m – (optional) measure value at the end of the curve

4.2.12.1 Methods on ST_SpiralCurve

1) ST_SpiralCurve: returns an ST_SpiralCurve value constructed from either:
   a) the well-known text representation of an ST_SpiralCurve value;
   b) the well-known binary representation of an ST_SpiralCurve value;
   c) the GML representation of an ST_SpiralCurve value;

      NOTE There is no normative GML type as of GML version 3.2.1. One has formally been proposed.
   d) the specified ST_AffinePlacement reference location, DOUBLE PRECISION length, start and end curvature and CHARACTER VARYING spiral type values;
   e) the specified ST_AffinePlacement reference location, DOUBLE PRECISION length, start and end curvature, CHARACTER VARYING spiral type and DOUBLE PRECISION start and end measure values.
2) ST_RefLocation: observes and mutates the ST_AffinePlacement reference location value in the ST_SpiralCurve value.
3) ST_Length: observes and mutates the DOUBLE PRECISION length value of an ST_SpiralCurve value.
4) ST_StartCurvature: observes and mutates the DOUBLE PRECISION start curvature value of an ST_SpiralCurve value.
5) ST_EndCurvature: observes and mutates the DOUBLE PRECISION end curvature value of an ST_SpiralCurve value.
6) ST_SpiralType returns the type of the ST_SpiralCurve value as a CHARACTER VARYING value.
7) ST_StartM: observes and mutates the measure value at the start of an ST_SpiralCurve value.
8) ST_EndM: observes and mutates the measure value at the end of an ST_SpiralCurve value.

4.2.12.2 Functions on ST_SpiralCurve

1) ST_SpiralFromTxt: returns an ST_SpiralCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_SpiralCurve.
2) ST_SpiralFromWKB: returns an ST_SpiralCurve value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_SpiralCurve.
3) ST_SpiralFromGML: returns an ST_SpiralCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_SpiralCurve value.

4.2.13 ST_CompoundCurve

The ST_CompoundCurve type is a subtype of ST_Curve. The ST_CompoundCurve type is instantiable. The general notion of a compound curve is a sequence of contiguous curves such that adjacent curves are joined at their end points. The end point of each curve shall be equal to the start point of the next curve in the list.

4.2.13.1 Methods on ST_CompoundCurve

1) ST_CompoundCurve: returns an ST_CompoundCurve value constructed from either:
   a) the well-known text representation of an ST_CompoundCurve value;
   b) the well-known binary representation of an ST_CompoundCurve value;
   c) the GML representation of an ST_CompoundCurve value;
   d) the specified ST_Curve values.
2) ST_Curves: observes and mutates the ST_Curve collection in the ST_CompoundCurve value.
3) ST_NumCurves: returns the cardinality of the ST_Curve collection in the ST_CompoundCurve value.
4) ST_CurveN: returns the specified element in the ST_Curve collection in the ST_CompoundCurve value.

4.2.13.2 Functions on ST_CompoundCurve

1) ST_CompoundFromTxt: returns an ST_CompoundCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CompoundCurve.
2) ST_CompoundFromWKB: returns an ST_CompoundCurve value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CompoundCurve.
3) ST_CompoundFromGML: returns an ST_CompoundCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_CompoundCurve value.

4.2.14 ST_Surface

The ST_Surface type is a subtype of ST_Geometry. The ST_Surface type is not instantiable. An ST_Surface value is a 2-dimensional geometry that consists of a single connected interior that is associated with one exterior ring and zero or more interior rings. Surfaces in three-dimensional coordinate space are isomorphic to planar surfaces. Polyhedral surfaces are formed by stitching together simple surfaces along their boundaries, Polyhedral surfaces in three-dimensional coordinate space may not be planar.

The boundary of a simple surface is the set of closed curves corresponding to its exterior and interior rings. A simple surface representing a single component consisting of any number of surfaces connected in a topological cycle and whose boundary is empty is called a shell. Unlike the curves in a ring, which is also simple and closed, the surfaces in a shell have no natural sort order.

4.2.14.1 Methods on ST_Surface

1) ST_Area: returns the area of an ST_Surface value, ignoring z and m coordinate values in the calculations.

2) ST_3DArea: returns the area of an ST_Surface value, considering z coordinate values in the calculations.
3) ST_Perimeter: returns the length of the boundary of an ST_Surface value, ignoring z and m coordinate values in the calculations.
4) ST_3DPerimeter: returns the length of the boundary of an ST_Surface value, considering z coordinate values in the calculations.
5) ST_Centroid: returns the ST_Point value that is the mathematical centroid of the ST_Surface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
6) ST_3DCentroid: returns the ST_Point value that is the mathematical centroid of the ST_Surface value, considering z coordinate values in the calculations and including them in the resultant geometry.
7) ST_PointOnSurface: returns the ST_Point value that is guaranteed to intersect the ST_Surface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
8) ST_3DPointOnSurf: returns the ST_Point value that is guaranteed to intersect the ST_Surface value, considering z coordinate values in the calculations and including them in the resultant geometry.
9) ST_IsWorld: test if the exterior of the ST_Surface value is the empty set, ignoring z and m coordinate values in the calculations.
10) ST_Is3DClosed: tests if an ST_Surface value is closed, considering z (but not m) coordinate values in the calculations.
11) ST_IsShell: test if the ST_Surface value is a shell, considering z (but not m) coordinate values in the calculations.

4.2.15 ST_CurvePolygon

The ST_CurvePolygon type is a subtype of ST_Surface. The ST_CurvePolygon type is instantiable. An ST_CurvePolygon value is a planar surface consisting of a single patch, defined by one exterior boundary and zero or more interior boundaries. Each interior boundary defines a hole in the ST_CurvePolygon value.

ST_CurvePolygon values are topologically closed. The boundary of an ST_CurvePolygon consists of an exterior ring and zero or more interior rings. No two rings in the boundary cross. The rings in the boundary of an ST_CurvePolygon value may intersect at a point. An ST_CurvePolygon shall not have cut lines, spikes or punctures. The interior of every ST_CurvePolygon is a connected point set. The exterior of an ST_CurvePolygon with one or more holes is not connected. Each hole defines a disconnected component of the exterior.

ST_CurvePolygon values are simple.

4.2.15.1 Methods on ST_CurvePolygon

1) ST_CurvePolygon: returns an ST_CurvePolygon value constructed from either:
   a) the well-known text representation of an ST_CurvePolygon value;
   b) the well-known binary representation of an ST_CurvePolygon value;
   c) the GML representation of an ST_CurvePolygon value;
   d) the specified ST_Curve values.
2) ST_ExteriorRing: observes and mutates the exterior ring of an ST_CurvePolygon value.
3) ST_InteriorRings: observes and mutates the collection of interior rings of an ST_CurvePolygon value.
4) ST_NumInteriorRing: returns the cardinality of the collection of interior rings of an ST_CurvePolygon value.

5) ST_InteriorRingN: returns the specified element in the collection of interior rings of an ST_CurvePolygon value.
6) ST_CurvePolyToPoly: returns an ST_Polygon value approximating the ST_CurvePolygon value, considering z and m coordinate values in the calculations and including z and m coordinate values in the resultant geometry.

4.2.15.2 Functions on ST_CurvePolygon

1) ST_CPolyFromText: returns an ST_CurvePolygon value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CurvePolygon.
2) ST_CPolyFromWKB: returns an ST_CurvePolygon value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CurvePolygon.
3) ST_CPolyFromGML: returns an ST_CurvePolygon value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Polygon or PolygonPatch representation of an ST_CurvePolygon value.

4.2.16 ST_Polygon

The ST_Polygon type is a subtype of ST_CurvePolygon whose boundary is defined by linear rings. The ST_Polygon type is instantiable.

4.2.16.1 Methods on ST_Polygon

1) ST_Polygon: returns an ST_Polygon value constructed from either:
   a) the well-known text representation of an ST_Polygon value;
   b) the well-known binary representation of an ST_Polygon value;
   c) the GML representation of an ST_Polygon value;
   d) the specified ST_LineString values.

4.2.16.2 Functions on ST_Polygon

1) ST_PolyFromText: returns an ST_Polygon value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Polygon.
2) ST_PolyFromWKB: returns an ST_Polygon value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Polygon.
3) ST_PolyFromGML: returns an ST_Polygon value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Polygon or PolygonPatch representation of an ST_Polygon value.
4) ST_BdPolyFromText: returns an ST_Polygon value, which is built from a well-known text representation of an ST_MultiLineString.
5) ST_BdPolyFromWKB: returns an ST_Polygon value, which is built from a well-known binary representation of an ST_MultiLineString.

4.2.17 ST_Triangle

The ST_Triangle type is a subtype of ST_Polygon with an exterior boundary having exactly four points (the last point being the same as the first point) and no interior boundaries. The ST_Triangle type is instantiable.

4.2.17.1 Methods on ST_Triangle

1) ST_Triangle: returns an ST_Triangle value constructed from either:
   a) the well-known text representation of an ST_Triangle value;
   b) the well-known binary representation of an ST_Triangle value;
   c) the GML representation of an ST_Triangle value;
   d) the specified ST_LineString value;

e) the specified ST_Point values.

2) ST_Points: observes and mutates the four ST_Points in the ST_LineString exterior boundary of the ST_Triangle value.

3) ST_3DSlope: returns the slope of a triangle value.

4.2.17.2 Functions on ST_Triangle

1) ST_TriFromText: returns an ST_Triangle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Triangle.

2) ST_TriFromWKB: returns an ST_Triangle value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Triangle.

3) ST_TriFromGML: returns an ST_Triangle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Triangle value.

4.2.18 ST_PolyhdrlSurface

The ST_PolyhdrlSurface type is a subtype of ST_Surface composed of contiguous polygon surfaces (ST_Polygon) connected along their common boundary curves. The ST_PolyhdrlSurface type is instantiable.

4.2.18.1 Methods on ST_PolyhdrlSurface

1) ST_PolyhdrlSurface: returns an ST_PolyhdrlSurface value constructed from either:
   a) the well-known text representation of an ST_PolyhdrlSurface value;
   b) the well-known binary representation of an ST_PolyhdrlSurface value;
   c) the GML representation of an ST_PolyhdrlSurface value;
   d) the specified ST_Polygon values.

2) ST_Patches: observes and mutates the ST_Polygon collection in the ST_PolyhdrlSurface value.

3) ST_NumPatches: returns the cardinality of the ST_Polygon collection in the ST_PolyhdrlSurface value.

4) ST_PatchN: returns the specified element in the ST_Polygon collection in the ST_PolyhdrlSurface value.

4.2.18.2 Functions on ST_PolyhdrlSurface

1) ST_PhSFromText: returns an ST_PolyhdrlSurface value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_PolyhdrlSurface.

2) ST_PhSFromWKB: returns an ST_PolyhdrlSurface value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_PolyhdrlSurface.

3) ST_PhSFromGML: returns an ST_PolyhdrlSurface value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML PolyhedralSurface or PolygonPatch representation of an ST_PolyhdrlSurface value.

4.2.19 ST_TIN

The ST_TIN type is a subtype of ST_PolyhdrlSurface composed only of triangles (ST_Triangle) that uses the Delaunay algorithm [3], or a similar implementation-defined algorithm, complemented with consideration for breaklines, soft breaks, control contours, break voids, drape voids, voids, holes, stop lines and maximum length of triangle sides. The ST_TIN type is instantiable.

4.2.19.1 Methods on ST_TIN

1) ST_TIN: returns an ST_TIN value constructed from either:
   a) the well-known text representation of an ST_TIN value;
   b) the well-known binary representation of an ST_TIN value;

c) the GML representation of an ST_TIN value;

d) the specified triangles (ST_Triangle values), TIN elements (ST_TINElement ARRAY value) and the DOUBLE PRECISION maximum allowable triangle side length;

e) the specified TIN elements (ST_TINElement ARRAY value) and the DOUBLE PRECISION maximum allowable triangle side length.

2) ST_TINElements: observes and mutates the ST_TINElement ARRAY collection of TIN elements in the ST_TIN value.
3) ST_MaxSideLength: observes and mutates the DOUBLE PRECISION maximum allowable triangle side length in the ST_TIN value.
4) ST_TINTable: observes and mutates the ST_TIN value in table format with point references.
5) ST_Clip: returns that part of an ST_TIN value that is within the clipping boundary.
6) ST_Patches: observes and mutates the ST_Triangle collection in the ST_TIN value.

4.2.19.2 Functions on ST_TIN

1) ST_TINFromText: returns an ST_TIN value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_TIN.
2) ST_TINFromWKB: returns an ST_TIN value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_TIN.
3) ST_TINFromGML: returns an ST_TIN value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML 3.2.1 or 3.3 representation of an ST_TIN value.

4.2.20 ST_CompoundSurface

The ST_CompoundSurface type is a subtype of ST_Surface. The ST_CompoundSurface type is instantiable. The general notion of a compound surface is a collection of surfaces that join in pairs on common boundary Surfaces and which, when considered as a whole, form a single surface.

4.2.20.1 Methods on ST_CompoundSurface

1) ST_CompoundSurface: returns an ST_CompoundSurface value constructed from either:

a) the well-known text representation of an ST_CompoundSurface value;

b) the well-known binary representation of an ST_CompoundSurface value;

c) the GML representation of an ST_CompoundSurface value;

d) the specified ST_Surface values.

2) ST_Surfaces: observes and mutates the ST_Surface collection in the ST_CompoundSurface value.
3) ST_NumSurfaces: returns the cardinality of the ST_Surface collection in the ST_CompoundSurface value.
4) ST_SurfaceN: returns the specified element in the ST_Surface collection in the ST_CompoundSurface value.

4.2.20.2 Functions on ST_CompoundSurface

1) ST_CompSurfFromTxt: returns an ST_CompoundSurface value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CompoundSurface.
2) ST_CompSurfFromWKB: returns an ST_CompoundSurfacevalue, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CompoundSurface.
3) ST_CompSurfFromGML: returns an ST_CompoundSurface value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_CompoundSurface.

4.2.21 ST_Solid

The ST_Solid type is a subtype of ST_Geometry. The ST_Solid type is not instantiable. An ST_Solid value is a 3-dimensional geometry representing the continuous image of a region of Euclidean 3 space.

4.2.21.1 Methods on ST_Solid

1) ST_3DSurfaceArea: returns the sum of the surface areas of all of the boundary components of a solid, considering z coordinate values in the calculations.
2) ST_3DVolume: returns the volume of this ST_Solid value which is the volume interior to the exterior boundary shell minus the sum of the volumes interior to any interior boundary shell. Z coordinates are considered in the calculations.
3) ST_3DCentroid: returns the ST_Point value that is the mathematical centroid of the ST_Solid value, considering z coordinate values in the calculations and including them in the resultant geometry.
4) ST_3DPointOnSolid: returns an ST_Point value guaranteed to spatially intersect the ST_Solid value, considering z coordinate values in the calculations and including them in the resultant geometry.

4.2.22 ST_BRepSolid

The ST_BRepSolid type is a subtype of ST_Solid. The ST_BRepSolid type is instantiable. An ST_BRepSolid value is a 3-dimensional geometry that consists of a single connected interior that is associated with one exterior shell and zero or more interior shells.

The boundary of a simple Brep solid is the set of closed surfaces corresponding to its exterior and interior shells.

4.2.22.1 Methods on ST_BRepSolid

1) ST_BRepSolid: returns an ST_BRepSolid value constructed from either:
   a) the well-known text representation of an ST_BRepSolid value;
   b) the well-known binary representation of an ST_BRepSolid value;
   c) the GML representation of an ST_BRepSolid value;
   d) the specified ST_Surface values.
2) ST_ExteriorShell: observes and mutates the exterior shell of an ST_BRepSolid value.
3) ST_InteriorShells: observes and mutates the collection of interior shells of an ST_BRepSolid value.
4) ST_NumIntShells: returns the cardinality of the collection of interior shells of an ST_BRepSolid value.
5) ST_InteriorShellN: returns the specified element in the collection of interior shells of an ST_BRepSolid value.

4.2.22.2 Functions on ST_BRepSolid

1) ST_BRepFromText: returns an ST_BRepSolid value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_BRepSolid.
2) ST_BRepFromWKB: returns an ST_BRepSolid value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_BRepSolid.
3) ST_BRepFromGML: returns an ST_BRepSolid value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_BRepSolid value.

4.2.23 ST_GeomCollection

The ST_GeomCollection type is a subtype of ST_Geometry. The ST_GeomCollection type is instantiable. An ST_GeomCollection is a collection of zero or more ST_Geometry values.

All the elements in an ST_GeomCollection are in the same spatial reference system. This is also the spatial reference system for the ST_GeomCollection value.

The ST_GeomCollection type places no other constraints on its elements. Subtypes of ST_GeomCollection may restrict membership based on dimension or place other constraints on the degree of spatial overlap between elements.

4.2.23.1 Methods on ST_GeomCollection

1) ST_GeomCollection: returns an ST_GeomCollection value constructed from either:
   a) the well-known text representation of an ST_GeomCollection value;
   b) the well-known binary representation of an ST_GeomCollection value;
   c) the GML representation of an ST_GeomCollection value;
   d) the specified ST_Geometry values.
2) ST_Geometries: observes and mutates the ST_Geometry collection in the ST_GeomCollection value.
3) ST_NumGeometries: returns the cardinality of the ST_Geometry collection in the ST_GeomCollection value.
4) ST_GeometryN: returns the specified element in the ST_Geometry collection in the ST_GeomCollection value.

4.2.23.2 Functions on ST_GeomCollection

1) ST_GeomCollFromTxt: returns an ST_GeomCollection value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_GeomCollection.
2) ST_GeomCollFromWKB: returns an ST_GeomCollection value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_GeomCollection.
3) ST_GeomCollFromGML: returns an ST_GeomCollection value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_GeomCollection value.

4.2.24 ST_MultiPoint

The ST_MultiPoint type is a subtype of ST_GeomCollection. The ST_MultiPoint type is instantiable. An ST_MultiPoint value is a 0-dimensional geometry. The elements of an ST_MultiPoint value are restricted to ST_Point values. The ST_Point values are not connected or ordered. An ST_MultiPoint value is simple if and only if no two ST_Point values in the ST_MultiPoint value are equal. The boundary of an ST_MultiPoint is the empty set.

4.2.24.1 Methods on ST_MultiPoint

1) ST_MultiPoint returns an ST_MultiPoint value constructed from either:
   a) the well-known text representation of an ST_MultiPoint value;
   b) the well-known binary representation of an ST_MultiPoint value;
   c) the GML representation of an ST_MultiPoint value;
   d) the specified ST_Point values.

4.2.24.2 Functions on ST_MultiPoint

1) ST_MPointFromText: returns an ST_MultiPoint value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiPoint.
2) ST_MPointFromWKB: returns an ST_MultiPoint value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiPoint.
3) ST_MPointFromGML: returns an ST_MultiPoint value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_MultiPoint value.

4.2.25 ST_MultiCurve

The ST_MultiCurve type is a subtype of ST_GeomCollection. The ST_MultiCurve type may be instantiable. An ST_MultiCurve is a 1-dimensional geometry. The elements of an ST_MultiCurve value are restricted to ST_Curve values.

An ST_MultiCurve is simple if and only if all of its elements are simple and the only intersections between any two elements occur at points that are in the boundaries of both elements. The boundary of an ST_MultiCurve is obtained by applying the mod 2 union rule: an ST_Point value is in the boundary of an ST_MultiCurve if it is in the boundaries of an odd number of elements of the ST_MultiCurve value.

An ST_MultiCurve value is closed if all of its elements are closed. The boundary of a closed ST_MultiCurve is the empty set. An ST_MultiCurve value is defined to be topologically closed.

4.2.25.1 Methods on ST_MultiCurve

1) ST_MultiCurve: returns an ST_MultiCurve value constructed from either:
    a) the well-known text representation of an ST_MultiCurve value;
    b) the well-known binary representation of an ST_MultiCurve value;
    c) the GML representation of an ST_MultiCurve value;
    d) the specified ST_Curve values.
2) ST_IsClosed: tests if an ST_MultiCurve value is closed, ignoring z and m coordinate values in the calculations.
3) ST_3DIsClosed: tests if an ST_MultiCurve value is closed, considering z coordinate values in the calculations.
4) ST_Length: returns the length of an ST_MultiCurve value, ignoring z and m coordinate values in the calculations.
5) ST_3DLength: returns the 3D length of the ST_MultiCurve value, considering z coordinate values in the calculations.
6) ST_PerpPoints: returns the geometry representing the perpendicular projection of the given point on the multicurve, ignoring z and m coordinate values in the calculations.

4.2.25.2 Functions on ST_MultiCurve

1) ST_MCurveFromText: returns an ST_MultiCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiCurve.
2) ST_MCurveFromWKB: returns an ST_MultiCurve value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiCurve.
3) ST_MCurveFromGML: returns an ST_MultiCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_MultiCurve value.

4.2.26 ST_MultiLineString

The ST_MultiLineString type is a subtype of ST_MultiCurve. The ST_MultiLineString type is instantiable. The elements of an ST_MultiLineString value are restricted to ST_LineString values.

4.2.26.1 Methods on ST_MultiLineString

1) ST_MultiLineString: returns an ST_MultiLineString value constructed from either:
    a) the well-known text representation of an ST_MultiLineString value;
    b) the well-known binary representation of an ST_MultiLineString value;
    c) the GML representation of an ST_MultiLineString value;
    d) the specified ST_LineString values.

4.2.26.2 Functions on ST_MultiLineString

1) ST_MLineFromText: returns an ST_MultiLineString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiLineString.
2) ST_MLineFromWKB: returns an ST_MultiLineString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiLineString.

3) ST_MLineFromGML: returns an ST_MultiLineString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_MultiLineString value.

4.2.27 ST_MultiSurface

The ST_MultiSurface type is a subtype of ST_GeomCollection. The ST_MultiSurface type may be instantiable. An ST_MultiSurface is a 2-dimensional geometry. The elements of an ST_MultiSurface value are restricted to ST_Surface values. The interiors of any two ST_Surface values in an ST_MultiSurface shall not intersect. The boundaries of any two elements in an ST_MultiSurface may intersect at a finite number of ST_Point values.

ST_MultiSurface values are simple.

4.2.27.1 Methods on ST_MultiSurface

1) ST_MultiSurface: returns an ST_MultiSurface value constructed from either:
   a) the well-known text representation of an ST_MultiSurface value;
   b) the well-known binary representation of an ST_MultiSurface value;
   c) the GML representation of an ST_MultiSurface value;
   d) the specified ST_Surface values.
2) ST_Area: returns the area of an ST_MultiSurface value, ignoring z and m coordinate values in the calculations.
3) ST_3DArea: returns the area of an ST_MultiSurface value, considering z coordinate values in the calculations.
4) ST_Perimeter: returns the length of the perimeter of an ST_MultiSurface value, ignoring z and m coordinate values in the calculations.
5) ST_3DPerimeter: returns the length of the perimeter of an ST_MultiSurface value, considering z coordinate values in the calculations.
6) ST_Centroid: returns the ST_Point value that is the mathematical centroid of the ST_MultiSurface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
7) ST_3DCentroid: returns the ST_Point value that is the mathematical centroid of the ST_MultiSurface value, considering z coordinate values in the calculations and including them in the resultant geometry.
8) ST_PointOnSurface: returns the ST_Point value that is guaranteed to intersect the ST_MultiSurface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
9) ST_3DPointOnSurf: returns the ST_Point value that is guaranteed to intersect the ST_MultiSurface value, considering z coordinate values in the calculations and including them in the resultant geometry.

4.2.27.2 Functions on ST_MultiSurface

1) ST_MSurfaceFromTxt: returns an ST_MultiSurface value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiSurface.
2) ST_MSurfaceFromWKB: returns an ST_MultiSurface value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiSurface.
3) ST_MSurfaceFromGML: returns an ST_MultiSurface value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_MultiSurface value.

4.2.28 ST_MultiPolygon

The ST_MultiPolygon type is a subtype of ST_MultiSurface. The ST_MultiPolygon type is instantiable. The elements of an ST_MultiPolygon value are restricted to ST_Polygon values. The interiors of any two ST_Polygon values that are elements of an ST_MultiPolygon shall not intersect. The boundaries of any two ST_Polygon values that are elements of an ST_MultiPolygon may touch at only a finite number of points.

An ST_MultiPolygon value shall not have cut lines, spikes or punctures. An ST_MultiPolygon value is a topologically closed point set. The interior of an ST_MultiPolygon value with more than one ST_Polygon value is not a connected point set. The number of disconnected components of the interior of an ST_MultiPolygon is equal to the number of ST_Polygon values in the ST_MultiPolygon. The boundary of an ST_MultiPolygon value is a set of linear rings corresponding to the boundaries of the ST_Polygon elements.

4.2.28.1 Methods on ST_MultiPolygon

1) ST_MultiPolygon: returns an ST_MultiPolygon value constructed from either:
   a) the well-known text representation of an ST_MultiPolygon value;
   b) the well-known binary representation of an ST_MultiPolygon value;
   c) the GML representation of an ST_MultiPolygon value;
   d) the specified ST_Polygon values.

4.2.28.2 Functions on ST_MultiPolygon

1) ST_MPolyFromText: returns an ST_MultiPolygon value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiPolygon.
2) ST_MPolyFromWKB: returns an ST_MultiPolygon value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiPolygon.
3) ST_MPolyFromGML: returns an ST_MultiPolygon value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_MultiPolygon value.
4) ST_BdMPolyFromText: returns an ST_MultiPolygon value, which is built from a well-known text representation of an ST_MultiLineString.
5) ST_BdMPolyFromWKB: returns an ST_MultiPolygon value, which is built from a well-known binary representation of an ST_MultiLineString.

4.3 Topology-Geometry

A topology-geometry (Topo-Geo) is a model assuming full planar topology, comprised of nodes, edges, and faces. Multiple, independent topology-geometries can be defined for different geographic areas or for different, overlapping sets of features in the same geographic area, as for land parcels and wetlands. Topology-geometries are distinguished by unique names. This topology-geometry name is specified herein as <topology-name>. A separate SQL-schema is used for each topology-geometry and is named <topology-name>.

The following views are defined for each <topology-name> SQL-schema: <topology-name>.ST_NODE, <topology-name>.ST_EDGE, and <topology-name>.ST_FACE.

The rows in these views define topological primitives of type node, edge, and face, respectively for the Topo-Geo called <topology-name>. Nodes and edges have associated ST_Geometry value. Faces may have a minimum bounding rectanglular geometry for spatial indexing. All geometry values for a given Topo-Geo shall have the same spatial reference system.

Each topological primitive has an ID, unique within the respective view, which allows the primitive to be referenced from another (e.g., feature or Topo-Geo) view.

The views provide the minimum information required to maintain a full planar topology. Support for additional information such as how these topological primitives are assigned to features or how real world attributes are assigned to topological primitives (e.g., edge weights) is not specified.

The ST_TOPO_GEO schema provides a suggested model of the base tables required to support the <topology-name> views. According to this model, all nodes from all topology-geometries reside in a single node base table, distinguished by a TOPOLOGY column containing <topology-name> values. Similarly, all edges reside in a single edge base table, and all faces reside in a single face base table, both with a distinguishing TOPOLOGY column.

A topology is topologically consistent if it exhibits the following characteristics:

1) all topological complexes are fully decomposed into their topological primitives
2) no two nodes exist at the same position in space
3) a node exists at the beginning and end of every edge
4) no edge has a geometry which crosses the geometry of a node
5) no edge has a geometry which crosses, overlaps, or is contained within the geometry of another edge
6) all edge geometries are simple
7) all edge geometries have a start point equal to the geometry of their start node
8) all edge geometries have a end point equal to the geometry of their end node
9) no face has a geometry which overlaps the geometry of another face
10) no face has a geometry within the geometry of another face
11) a universal face exists
12) a valid ST_Surface geometry can be constructed for all faces except the universal face
13) all geometries for the topology have the same spatial reference system

4.3.1 <topology-name>.ST_NODE

The <topology-name>.ST_NODE view contains the node type of topological primitives (ST_Node) contained in the <topology-name> topology-geometry. An ST_Node has a known not nullable, unique node ID of type integer and a known not nullable geometry of type ST_Point. If ST_Node is an isolated node, it has a containing face, identified by a <topology-name>.ST_FACE.ID.

4.3.1.1 Routines on <topology-name>.ST_NODE only

1) ST_AddIsoNode: for the provided topology-name, optional face ID, and ST_Point geometry, inserts a row into the <topology-name>.ST_NODE view corresponding to an isolated node, returning the generated unique integer node ID. If no face ID is provided, the function will determine which face the node will be within. If a face ID is provided, the ST_Point geometry shall be within the geometry of the face or an exception is raised. If another node in the <topology-name>.ST_NODE view exists at the ST_Point location or if the geometry of an existing edge crosses the ST_Point location, an exception is raised.
2) ST_MoveIsoNode: for the provided topology-name, node ID, and ST_Point geometry, updates the existing ST_Point geometry value. If the node is a connected node or if another node in the <topology-name>.ST_NODE view exists at the new location or it the geometry of an existing edge crosses the new ST_Point location, an exception is raised.
3) ST_RemIsoNode: deletes the row for the isolated node identified by the provided topology-name and node ID. If the node is a connected node, an exception is raised.

4.3.2 <topology-name>.ST_EDGE

The <topology-name>.ST_EDGE view contains the edge type of topological primitives (ST_Edge) contained in the <topology-name> Topology-geometry. An ST_Edge has a unique edge ID of type integer; node ID's of type integer for the start and end nodes; edge ID's of type integer for the next left face and next right face edges; face ID's of type integer for the left and right faces; and a geometry of type ST_Curve. All values are known not nullable. An isolated edge will have its containing face as both its left and right faces; the next right face edge will have an ID equal to the edge ID of the isolated edge and the next left face edge will have the negative of the isolated edge's ID.

Start and end node ID's are immutable. To change the start or end node of an edge, the edge shall be removed and a new one shall be created.

4.3.2.1 Routines on <topology-name>.ST_EDGE only

1) ST_AddIsoEdge: for the provided topology-name, start and end node IDs, and ST_Curve geometry, inserts a row into the <topology-name>.ST_EDGE view, returning the generated unique integer edge ID. The next right face edge is set equal to the new edge ID and the next left face edge is set equal to the negative of this value. The left and right face IDs are set equal to the containing face ID of the start and end node.

   An exception is raised for any of the following conditions:

   a) if the start and end node IDs do not correspond to existing isolated nodes,

   b) if the start and end node containing face IDs are not equal,

   c) if the ST_Point geometry of the start node does not equal the ST_StartPoint of the ST_Curve geometry of the edge,

   d) if the ST_Point geometry of the end node does not equal the ST_EndPoint of the ST_Curve geometry of the edge,

   e) if the ST_Curve geometry is not within the geometry of the containing face of the start and end nodes,

   f) if the ST_Curve geometry intersects the ST_Point geometry of any isolated node other than the start and end node,

   g) if the ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE view, or

   h) if the ST_Curve geometry is not simple.

2) ST_GetFaceEdges: for the provided topology-name and face ID, returns a table containing the integer edge IDs for the edges which bound the face, in counterclockwise order. Edge IDs will be negated in the query result if the face is right of the edge when looking in the direction of the edge from start to end node.

3) ST_ChangeEdgeGeom: for the provided topology-name, edge ID, and existing ST_Curve geometry, updates the ST_Curve geometry value.

   An exception is raised for any of the following conditions:

   a) if the ST_StartPoint of the new ST_Curve geometry is not equal to the ST_StartPoint of the existing ST_Curve geometry,

   b) if the ST_EndPoint of the new ST_Curve geometry is not equal to the ST_EndPoint of the existing ST_Curve geometry,

   c) if the interior of the new ST_Curve geometry intersects the ST_Point geometry of any isolated node in the <topology-name>.ST_NODE view,

   d) if the interior of the new ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE view, or

   e) if the ST_Curve geometry is not simple.

4) ST_RemIsoEdge: deletes the row for the isolated edge identified by the provided topology-name and edge ID. The start and end nodes are not removed from the <topology-name>.ST_NODE view. If the edge is a not an isolated edge, an exception is raised.

4.3.2.2 Routines on <topology-name>.ST_NODE and <topology-name>.ST_EDGE

1) ST_NewEdgesSplit: splits an edge by creating a new node along an existing edge, deleting the original edge and replacing it with two new edges. For the provided topology-name, edge ID, and ST_Point geometry,

   a) inserts a row into the <topology-name>.ST_NODE view, with geometry equal to the input ST_Point value,

   b) returns the generated unique integer node ID,

c) deletes the row in the <topology-name>.ST_EDGE view for the edge identified by the provided topology-name and edge ID, and

d) inserts two rows into the <topology-name>.ST_EDGE view for the two new resultant edges, deriving appropriate node, edge, and face values from the deleted edge,

e) creates ST_Curve geometries for the two new edges by splitting the geometry of the split edge at the ST_Point location,

f) makes any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edge being split.

To determine the two new edge IDs, query the <topology-name>.ST_EDGE view for edges with a start or end node equal to the returned node ID. Both new edges have the same direction as the edge being split. An exception is raised for any of the following conditions:

a) if the edge identified by the edge ID does not exist in the <topology-name>.ST_EDGE view,

b) if the ST_Point geometry is not within the ST_Curve geometry of the identified edge, or

c) if a node already exists in the <topology-name>.ST_NODE view at the input ST_Point geometry location.

2) ST_ModEdgeSplit: splits an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge. For the provided topology-name, edge ID, and ST_Point geometry,

a) inserts a row into the <topology-name>.ST_NODE view, with geometry equal to the input ST_Point value,

b) returns the generated unique integer node ID,

c) modifies the row in the <topology-name>.ST_EDGE view for the edge identified by the provided topology-name and edge ID, deriving appropriate node, edge, and face values from the original edge and new node,

d) inserts a new row into the <topology-name>.ST_EDGE view for the other new resultant edge, deriving appropriate node, edge, and face values from the original edge and new node,

e) creates ST_Curve geometries for the new and modified edges by splitting the geometry of the original edge at the ST_Point location,

f) makes any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edge being split.

To determine the new edge ID, query the <topology-name>.ST_EDGE view for the edge with a start node equal to the returned node ID. The new and modified edges have the same direction as the original edge. An exception is raised for any of the following conditions:

a) if the edge identified by the edge ID does not exist in the <topology-name>.ST_EDGE view,

b) if the ST_Point geometry is not within the ST_Curve geometry of the identified edge, or

c) if a node already exists in the <topology-name>.ST_NODE view at the input ST_Point geometry location.

3) ST_NewEdgeHeal: heals two edges by deleting the node connecting them, deleting both edges, and replacing them with a new edge whose direction is the same as the first edge provided. For the provided topology-name and two edge IDs,

a) deletes the row in the <topology-name>.ST_NODE view corresponding to the node shared by the two identified edges,

b) deletes the two rows in the <topology-name>.ST_EDGE view identified by the input edge IDs,

c) inserts a new row into the <topology-name>.ST_EDGE view for the resultant edge, deriving appropriate node, edge, and face values from the deleted edges,

d) creates an ST_Curve geometry for the new edge from the geometries of the two deleted edges,

e) returns the generated unique integer edge ID for the new edge, and

f) makes any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edges being healed.

The direction of the new edge shall be the same as the direction of the first supplied edge. An exception is raised for any of the following conditions:

a) if either edge identified by the edge IDs does not exist in the <topology-name>.ST_EDGE view,

b) if the two edges do not share a common node, or

c) if additional edges also share the common node.

4) ST_ModEdgeHeal: heals two edges by deleting the node connecting them, modifying the first edge provided, and deleting the second edge. For the provided topology-name and two edge IDs,

a) deletes the row in the <topology-name>.ST_NODE view corresponding to the node shared by the two identified edges,

b) deletes the row in the <topology-name>.ST_EDGE view identified by the second input edge ID,

c) modifies the values in the row in the <topology-name>.ST_EDGE view for the other edge, deriving appropriate node, edge, and face values from the original edges,

d) creates an ST_Curve geometry for the modified edge from the geometries of the two original edges,

e) makes any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edges being healed.

An exception is raised for any of the following conditions:

a) if either edge identified by the edge IDs does not exist in the <topology-name>.ST_EDGE view,

b) if the two edges do not share a common node, or

c) if additional edges also share the common node.

4.3.3 <topology-name>.ST_FACE

The <topology-name>.ST_FACE view contains the face type of topological primitives (ST_Face) contained in the <topology-name> Topology-geometry. An ST_Face has a known not nullable, unique face ID of type integer and a possibly nullable MBR (minimum bounding rectangle) geometry of type ST_Polygon.

The <topology-name>.ST_FACE view contains a row for the universal face. The universal face contains everything else in the topology exterior to all other faces. This face has a face ID = 0 (zero). There is no geometry associated with the universal face.

4.3.3.1 Routines on <topology-name>.ST_EDGE and <topology-name>.ST_FACE

1) ST_AddEdgeNewFaces: adds a new edge and, if in doing so it splits a face, deletes the original face and replaces it with two new faces. For the provided topology-name, start and end node IDs, and ST_Curve geometry,

a) inserts a row into the <topology-name>.ST_EDGE view, with start and end nodes as specified, automatically determined next edges and left and right faces, and geometry equal to the input ST_Curve value,

b) returns the generated unique integer edge ID, and

c) if the new edge splits a face, then

i) deletes the row in the <topology-name>.ST_FACE view corresponding to the face being split,

ii) automatically generates two new unique integer face IDs,

iii) inserts two rows into the <topology-name>.ST_FACE view for the two new resultant faces,

iv) creates ST_Polygon MBR geometries for the two new faces, and

v) updates the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split.

To determine the two new face IDs, query the <topology-name>.ST_EDGE view for the left and right faces for the edge with the returned edge ID. An exception is raised for any of the following conditions:

a) if either the start or end nodes identified do not exist in the <topology-name>.ST_NODE view,

b) if the ST_StartPoint of the new ST_Curve geometry is not equal to the ST_Point value of the start node geometry,

c) if the ST_EndPoint of the new ST_Curve geometry is not equal to the ST_Point value of the end node geometry,

d) if the interior of the new ST_Curve geometry intersects the ST_Point geometry of any isolated node in the <topology-name>.ST_NODE view,

e) if the interior of the new ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE view,

f) if an edge already exists in the <topology-name>.ST_EDGE view with the same terminal nodes and geometry, or

g) if the ST_Curve geometry is not simple.

2) ST_AddEdgeModFace: adds a new edge and if in doing so it splits a face, modifies the original face and adds a new face. For the provided topology-name, start and end node IDs, and ST_Curve geometry,

a) inserts a row into the <topology-name>.ST_EDGE view, with start and end nodes as specified, automatically determined next edges and left and right faces, and geometry equal to the input ST_Curve value,

b) returns the generated unique integer edge ID, and

c) if the new edge splits a face, then

i) modifies the ST_Polygon MBR geometry in the <topology-name>.ST_FACE view for the face being split,

ii) inserts a new row into the <topology-name>.ST_FACE view for the other new resultant face,

iii) creates an ST_Polygon MBR geometry for the new face, and

iv) updates the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split.

To determine the new and modified face IDs, query the <topology-name>.ST_EDGE view for the left and right faces for the edge with the returned edge ID. An exception is raised for any of the following conditions:

a) if either the start or end nodes identified do not exist in the <topology-name>.ST_NODE view,

b) if the ST_StartPoint of the new ST_Curve geometry is not equal to the ST_Point value of the start node geometry,

c) if the ST_EndPoint of the new ST_Curve geometry is not equal to the ST_Point value of the end node geometry,

d) if the interior of the new ST_Curve geometry intersects the ST_Point geometry of any isolated node in the <topology-name>.ST_NODE view,

e) if the interior of the new ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE view,

f) if an edge already exists in the <topology-name>.ST_EDGE view with the same terminal nodes and geometry, or

g) if the ST_Curve geometry is not simple.

3) ST_RemEdgeNewFace: removes an edge and, if the removed edge separated two faces, deletes the original faces and replaces them with one new face. For the provided topology-name and edge ID,

a) deletes the row in the <topology-name>.ST_EDGE view identified by the edge ID, and

b) if the edge removal results in the healing of two faces, then

i) deletes the two rows in the <topology-name>.ST_FACE view corresponding to the faces being healed,

ii) inserts a new row into the <topology-name>.ST_FACE view for the new resultant face,

iii) creates an ST_Polygon MBR geometry for the new face,

iv) returns the generated unique integer face ID, and

v) updates the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed.

The start and end nodes of the deleted edge remain in the <topology-name>.ST_NODE view. An exception is raised if the edge identified by the edge ID does not exist in the <topology-name>.ST_EDGE view.

4) ST_RemEdgeModFace: removes an edge and, if the removed edge separated two faces, heals the two faces by modifying one of the faces and deleting the other. For the provided topology-name and edge ID,

a) deletes the row in the <topology-name>.ST_EDGE view identified by the edge ID, and

b) if the edge removal results in the healing of two faces, then

i) deletes the row in the <topology-name>.ST_FACE view corresponding to one of the faces being healed,

ii) creates a new ST_Polygon MBR geometry for the modified face,

iii) updates the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed.

The choice of which face to modify and which to delete is implementation-dependent.

The start and end nodes of the deleted edge remain in the <topology-name>.ST_NODE view. An exception is raised if the edge identified by the edge ID does not exist in the <topology-name>.ST_EDGE view.

5) ST_GetFaceGeometry: for the provided topology-name and face ID, returns the exact geometry of the face:

a) determines the edges in the <topology-name>.ST_Edge view which bound the face identified by the face ID

b) retrieves the ST_Curve geometries from the <topology-name>.ST_Edge view for each of these edges

c) returns a valid ST_Surface geometry value constructed from the edge geometries, if possible, or else an empty set of type ST_Surface.

An exception is raised if the face identified by the face ID does not exist in the <topology-name>.ST_FACE view, or if it is the universal face.

4.3.3.2 Routines on <topology-name>.ST_NODE, <topology-name>.ST_EDGE and <topology-name>.ST_FACE

1) ST_InitTopoGeo: for the provided topology-name, creates the <topology-name> schema, the ST_NODE, ST_EDGE, and ST_FACE views for this schema, and a row in the ST_FACE view contains the universal face. An exception is raised if a schema already exists with that name.

2) ST_CreateTopoGeo: for the provided topology-name, and ST_GeomCollection, populates the <topology-name>.ST_NODE, <topology-name>.ST_EDGE, and <topology-name>.ST_FACE views from the geometry values in the ST_GeomCollection. An exception is raised if any of these three views do not already exist or if they already contain any rows other than one for the universal face.

3) ST_ValidateTopoGeo: for the provided topology-name, returns a table containing possible topological inconsistencies.

4.4 Topology-Network

A topology-network (Topo-Net) is a model for linear applications including node and link topological primitives. Multiple, independent topology-networks can be defined for different geographic areas or for different, overlapping sets of features in the same geographic area, as for road and rail networks. Topology-networks are distinguished by unique names. This topology-network name is specified herein as <network-name>. A separate SQL-schema is used for each topology-network and is named <network-name>.

The following views are defined for each <network-name> SQL-schema: <network-name>.ST_NODE and <network-name>.ST_LINK.

The rows in these views define network primitives of type node and link, respectively for the Topo-Net called <network-name>. These network primitives may have associated ST_Geometry values. All geometry values for a given Topo-Net shall have the same spatial reference system.

Each topological primitive has an ID, unique within the respective view, which allows the primitive to be referenced from another (e.g., feature or Topo-Net) view.

The views provide the minimum information required to maintain a linear network. Support for additional information such as how these topological primitives are assigned to features or how real world attributes are assigned to topological primitives (e.g., link weights) is not specified.

The ST_TOPO_NET schema provides a suggested model of the base tables required to support the <network-name> views. According to this model, all nodes from all topology-networks reside in a single node base table, distinguished by a NETWORK column containing <network-name> values. Similarly, all links reside in a single link base table with a distinguishing NETWORK column.

4.4.1 <network-name>.ST_NODE

The <network-name>.ST_NODE view contains the node type of network primitives (ST_Node) contained in the <network-name> topology-network. An ST_Node has a known not nullable, unique node ID of type integer and a possibly nullable geometry of type ST_Point.

4.4.1.1 Routines on <network-name>.ST_NODE only

1) ST_AddIsoNetNode: for the provided network-name, and optional ST_Point geometry, inserts a row into the <network-name>.ST_NODE view, returning the generated unique integer node ID.
2) ST_MoveIsoNetNode: for the provided network-name, node ID, and ST_Point geometry, updates the ST_Point geometry value. If the node is a connected node exception is raised.
3) ST_RemIsoNetNode: deletes the row for the isolated node identified by the provided network-name and node ID. If the node is a connected node, an exception is raised.

4.4.2 <network-name>.ST_LINK

The <network-name>.ST_LINK view contains the link type of network primitives (ST_Link) contained in the <network-name> topology-network. An ST_Link has a known not nullable unique link ID of type integer; known not nullable node IDs of type integer for the start and end nodes; and a possibly nullable geometry of type ST_Curve.

Start and end node ID's are immutable. To change the start or end node of an link, the link shall be removed and a new one shall be created.

4.4.2.1 Routines on <network-name>.ST_LINK only

1) ST_AddLink: for the provided network-name, start and end node IDs, and optional ST_Curve geometry, inserts a row into the <network-name>.ST_LINK view, returning the generated unique integer link ID.

   An exception is raised for any of the following conditions:

   a) if the start and end nodes are not existing nodes in the <network-name>.ST_NODE view,
   b) if a non-null ST_Curve geometry value is provided, then
      i) if the start node has a geometry and the location thereby specified does not equal the location of the ST_StartPoint of the proposed ST_Curve geometry of the link, or

ii) if the end node has a geometry and the location thereby specified does not equal the location of the ST_EndPoint of the proposed ST_Curve geometry of the link.

2) ST_ChangeLinkGeom: for the provided network-name, link ID, and ST_Curve geometry, updates the ST_Curve geometry value.

   An exception is raised if a non-null ST_Curve geometry value is provided and any of the following are true:

   a) the start node of the specified link has a null geometry value,

   b) the start node of the specified link has a non-null geometry value and this location is not equal to the location of the start point of the ST_Curve value,

   c) the end node of the specified link has a null geometry value, or

   d) the end node of the specified link has a non-null geometry value and this location is not equal to the location of the end point of the ST_Curve value.

3) ST_RemoveLink: deletes the row for the link identified by the provided network-name and link ID. The start and end nodes are not removed from the <network-name>.ST_NODE view.

4.4.2.2 Routines on <network-name>.ST_NODE and <network-name>.ST_LINK

1) ST_InitTopoNet: for the provided network-name, creates the <network-name> schema and the ST_NODE and ST_LINK views for this schema. An exception is raised if a schema already exists with that name.

2) ST_NewLogLinkSplit: splits a link in a logical network by creating a new node along an existing link, deleting the original link and replacing it with two new links. For the provided network-name and link ID,

   a) inserts a row into the <network-name>.ST_NODE view, with null values for Geometry,

   b) returns the generated unique integer node ID,

   c) deletes the row in the <network-name>.ST_LINK view for the link identified by the provided network-name and link ID, and

   d) inserts two rows into the <network-name>.ST_LINK view for the two new resultant links, deriving appropriate start and end node IDs from the deleted link and newly generated node.

   To determine the two new link IDs, query the <network-name>.ST_LINK view for links with a start or end node equal to the returned node ID. Both new links have the same direction as the link being split. An exception is raised if the link identified by the link ID does not exist in the <network-name>.ST_LINK view,

3) ST_ModLogLinkSplit: splits a logical network link by creating a new node along an existing link, modifying the original link and adding a new link. For the provided network-name and link ID,

   a) inserts a row into the <network-name>.ST_NODE view, with null values for Geometry,

   b) returns the generated unique integer node ID,

   c) modifies the row in the <network-name>.ST_LINK view for the link identified by the provided network-name and link ID, deriving appropriate start and end node IDs from the original link and newly generated node, and

   d) inserts a new row into the <network-name>.ST_LINK view for the other new resultant link, deriving appropriate start and end node IDs from the original link and newly generated node.

   To determine the new link ID, query the <network-name>.ST_LINK view for the link with a start node equal to the returned node ID. The new and modified links have the same direction as the link being split. An exception is raised if the link identified by the link ID does not exist in the <network-name>.ST_LINK view,

4) ST_NewGeoLinkSplit: splits a link in a network with geometry by creating a new node along an existing link, deleting the original link and replacing it with two new links. For the provided network-name, link ID, and ST_Point geometry,

a) inserts a row into the <network-name>.ST_NODE, with geometry equal to the input ST_Point value,

b) returns the generated unique integer node ID,

c) deletes the row in the <network-name>.ST_LINK for the link identified by the provided network-name and link ID,

d) inserts two rows into the <network-name>.ST_LINK for the two new resultant links, deriving appropriate start and end node IDs from the deleted link and newly generated node, and

e) creates ST_Curve geometries for the two new links by splitting the geometry of the split link at the ST_Point location.

To determine the two new link IDs, query the <network-name>.ST_LINK view for the link with a start node equal to the returned node ID. Both new links have the same direction as the link being split. An exception is raised for any of the following conditions:

a) if the link identified by the link ID does not exist in the <network-name>.ST_LINK,

b) if the ST_Point geometry is not within the ST_Curve geometry of the identified link,

c) if the link identified by the link ID contains a null geometry value.

5) ST_ ModGeoLinkSplit: splits a link in a network with geometry by creating a new node along an existing link, modifying the original link and adding a new link. For the provided network-name and link ID,

a) inserts a row into the <network-name>.ST_NODE view, with geometry equal to the input ST_Point value,

b) returns the generated unique integer node ID,

c) modifies the row in the <network-name>.ST_LINK view for the link identified by the provided network-name and link ID, deriving appropriate start and end node IDs from the original link and newly generated node,

d) inserts a new row into the <network-name>.ST_LINK view for the other new resultant link, deriving appropriate start and end node IDs from the original link and newly generated node, and

e) creates ST_Curve geometries for the new and modified links by splitting the geometry of the split link at the ST_Point location.

To determine the new link ID, query the <network-name>.ST_LINK view for links with a start or end node equal to the returned node ID. The new and modified links have the same direction as the link being split.

An exception is raised for any of the following conditions:

a) if the link identified by the link ID does not exist in the <network-name>.ST_LINK table,

b) if the ST_Point geometry is not within the ST_Curve geometry of the identified link,

c) if the link identified by the link ID contains a null geometry value.

6) ST_NewLinkHeal: heals two links by deleting the node connecting them, deleting both links, and replacing them with a new link, whose direction is the same as the first link provided. For the provided network-name and two link IDs,

a) if no other links start or end at the node shared by the two identified links, deletes the row in the <network-name>.ST_NODE view corresponding to the node shared by the two identified links,

b) deletes the two rows in the <network-name>.ST_LINK view identified by the input link IDs,

c) inserts a new row into the <network-name>.ST_LINK view for the resultant link, deriving appropriate start and end node IDs from the deleted links,

d) if the two links have geometry, creates an ST_Curve geometry for the new link from the geometries of the two deleted links,

e) returns the generated unique integer link ID for the new link.

The direction of the new link shall be the same as the direction of the first supplied link. An exception is raised for any of the following conditions:

a) if either link identified by the link IDs does not exist in the <network-name>.ST_LINK view, or

b) if the two links do not share a common node.

7) ST_ModLinkHeal: heals two links by deleting the node connecting them, modifying the first link and deleting the other. For the provided network-name and two link IDs,

a) if no other links start or end at the node shared by the two identified links, deletes the row in the <network-name>.ST_NODE view corresponding to the node shared by the two identified links,

b) deletes the row in the <network-name>.ST_LINK view identified by one of the input link IDs,

c) modifies the values in the row in the <network-name>.ST_LINK view for the other link, deriving appropriate start and end node IDs from the original links,

d) if the two links had geometry, creates a new ST_Curve geometry for the modified link from the geometries of the two original links.

The original direction of the modified link is retained.

An exception is raised for any of the following conditions:

a) if either link identified by the link IDs does not exist in the <network-name>.ST_LINK view, or

b) if the two links do not share a common node.

8) ST_LogiNetFromTGeo: for the provided network-name, and topology-name, creates a logical network by populating the <network-name>.ST_NODE and <network-name>.ST_LINK views from the Topo-Geo values identified by the provided topology-name. A Topo-Net node will be created for each Topo-Geo node. A Topo-Net link will be created for each Topo-Geo edge. A logical network will result, with nodes and links having geometry values set to the null value. An exception is raised if either of the two Topo-Net views do not already exist or if they already contain any rows or if any of the three Topo-Geo views do not exist.

9) ST_SpatNetFromTGeo: for the provided network-name, and topology-name, creates a spatial network by populating the <network-name>.ST_NODE and <network-name>.ST_LINK views from the Topo-Geo values identified by the provided topology-name. A Topo-Net node will be created for each Topo-Geo node. A Topo-Net link will be created for each Topo-Geo edge. A spatial network will result, with nodes and links having geometry values obtained from their corresponding Topo-Geo primitives. An exception is raised if either of the two Topo-Net views do not already exist or if they already contain any rows or if any of the three Topo-Geo views do not exist.

10) ST_SpatNetFromGeom: for the provided network-name, and ST_GeomCollection, creates a spatial network by populating the <network-name>.ST_NODE and <network-name>.ST_LINK views from the geometry values in the ST_GeomCollection. A node will be created wherever links start, end, or cross. A spatial network will result, with nodes and links having geometry values. An exception is raised if either of these two views do not already exist or if they already contain any rows.

11) ST_ValidLogicalNet: for the provided network-name, returns a table containing possible logical network inconsistencies.

12) ST_ValidSpatialNet: for the provided network-name, returns a table containing possible spatial network inconsistencies.

4.5 General Routines

4.5.1 ST_ShortestUndPath Function

A table function ST_ShortestUndPath calculates combinatorial geometric weighted distances between two specified points that are to be non-closed terminal points of an ST_Geometry value in a referenced table with undirected 1-dimensional simple geometry, and returns IDs of the shortest paths in form of a table.

4.5.2 ST_ShortestDirPath Function

A table function ST_ShortestDirPath calculates combinatorial geometric weighted distances between two specified points that are to be non-closed terminal points of an ST_Geometry value in a referenced table with directed 1-dimensional simple geometry, and returns IDs of the shortest paths in the form of a table.

4.6 Spatial Reference System Type

4.6.1 ST_SpatialRefSys

The ST_SpatialRefSys type encapsulates all aspects of spatial reference systems.

4.6.1.1 Methods on ST_SpatialRefSys

1) ST_SpatialRefSys: returns the specified ST_SpatialRefSys value constructed from either:
   a) the well-known text representation of spatial reference system;
   b) a spatial reference system identifier.
2) ST_AsWKTSRS: returns the well-known text representation of a spatial reference system for the specified ST_SpatialRefSys value.
3) ST_WKTSRSToSQL: returns the ST_SpatialRefSys value represented by the specified well-known text representation of a spatial reference system.
4) ST_SRID: returns the integer identifier of an ST_SpatialRefSys value.
5) ST_Equals: tests if two ST_SpatialRefSys values are equal.

4.6.1.2 Ordering on ST_SpatialRefSys

1) ST_OrderingEquals: is the equals only ordering definition for the ST_SpatialRefSys type.

4.6.1.3 SQL Transforms on ST_SpatialRefSys

1) ST_WellKnownText: is the SQL Transform group that transforms an ST_SpatialRefSys value to and from a well-known text representation of a spatial reference system in a CHARACTER LARGE OBJECT value.

4.7 Linear Referencing Types

The linear referencing types encapsulate aspects of linear referencing requisite for defining linearly referenced locations. The types derive from the Generalized Model for Linear Referencing [4] and are in accordance with the conceptual model in ISO 19148:2012.

The following types are supported: ST_PositionExp, ST_LinearElement, ST_LRFeature, ST_LRCurve, ST_LRDirectedEdge, ST_LRM, ST_DistanceExp, ST_LRMeasure, ST_StartValue, ST_Referent, ST_LatOffsetExp, ST_VerOffsetExp and ST_VectorOffsetExp.

All of these types except ST_LinearElement are instantiable and have explicitly defined constructor functions.

ST_PositionExp, ST_LinearElement, ST_LRFeature, ST_LRCurve, ST_LRDirectedEdge, ST_LRM and ST_DistanceExp can be used as the type of a column. ST_LRMeasure, ST_StartValue, ST_Referent, ST_LatOffsetExp, ST_VerOffsetExp and ST_VectorOffsetExp are only used as attributes of the forementioned types. Declaring a column to be of a particular type implies that any value of the type or any of its subtypes can be used.

4.7.1 ST_PositionExp

The ST_PositionExp type is used to specify a position as a linearly referenced location given by the linear element being measured, the method of measurement (LRM) and a measure value specified by a distance expression. The ST_PositionExp type is instantiable.

4.7.1.1 Methods on ST_PositionExp

1) ST_PositionExp: returns a specified ST_PositionExp value from either:
   a) the well-known text representation of an ST_PositionExp value;

b) the GML representation of an ST_PositionExp value;

c) the specified INTEGER linear element ID (leid), INTEGER Linear Referencing Method ID (lrmid) and ST_DistanceExp distance expression values.

d) the specified INTEGER linear element ID (leid), ST_LRM Linear Referencing Method and ST_DistanceExp distance expression values.

e) the specified ST_LinearElement linear element, INTEGER Linear Referencing Method ID (lrmid) and ST_DistanceExp distance expression values.

f) the specified ST_LinearElement linear element, ST_LRM Linear Referencing Method and ST_DistanceExp distance expression values.

2) ST_LinearElementID: observes and mutates the linear element ID (leid) value of the ST_PositionExp value.

3) ST_LinearElement: observes and mutates the linear element value of the ST_PositionExp value.

4) ST_LRMID: observes and mutates the Linear Referencing Method ID (lrmid) value of the ST_PositionExp value.

5) ST_LRM: observes and mutates the Linear Referencing Method value of the ST_PositionExp value.

6) ST_DistanceExp: observes and mutates the distance expression value of the ST_PositionExp value.

7) ST_Equals: tests if an ST_PositionExp specifies the same linearly referenced location as another ST_PositionExp value. This test is for equivalence between two, possibly quite different, representations.

4.7.1.2 Functions on ST_PositionExp

1) ST_PosExpFromText: returns an ST_PositionExp value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_PositionExp value.

2) ST_PosExpFromGML: returns an ST_PositionExp value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_PositionExp value.

4.7.2 ST_LinearElement

The ST_LinearElement type specifies the underlying linear element upon which the measures in the Linear Referencing System are made. The linear element can be either a feature, a curve geometry or a topological edge. The ST_LinearElement type is not instantiable: its subtypes are.

4.7.2.1 Methods on ST_LinearElement

1) ST_LinearElementID: observes and mutates the linear element ID (leid) value of the ST_LinearElement value.

2) ST_DefaultLRM: observes and mutates the default Linear Referencing Method lrmid value of the ST_LinearElement value.

3) ST_DefaultMeasure: observes and mutates the default length value of the ST_LinearElement value.

4) ST_LEType: observes and mutates the linear element type value of the ST_LinearElement value.

5) ST_StartValue: observes and mutates the measure value at the start of the ST_LinearElement for the specified Linear Referencing Method lrmid. This is usually 0 (zero).

6) ST_TranslateToInst: translates an ST_PositionExp defined along the subject (source) ST_LinearElement into an ST_DistanceExp measured along a known, specified target ST_LinearElement using the target Linear Referencing Method.

7) ST_TranslateToType: translates an ST_PositionExp defined along the subject (source) ST_LinearElement into one or more ST_PositionExps measured along the appropriate instances of the linear element type specified, using the target Linear Referencing Method.

4.7.2.2 Functions on ST_LinearElement

1) ST_LEFromText: returns an ST_LinearElement value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LinearElement value.
2) ST_LEFromGML: returns an ST_LinearElement value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LinearElement value.

4.7.3 ST_LRFeature

The ST_LRFeature subtype of ST_LinearElement specifies any feature which can be linearly measured, that is, which supports the methods of ST_LinearElement. The ST_LRFeature type is instantiable. The concept of “feature” derives from ISO 19109:2005 [5].

4.7.3.1 Methods on ST_LRFeature

1) ST_LRFeature: returns a specified ST_LRFeature value from either:
   a) the well-known text representation of an ST_LRFeature value;
   b) the GML representation of an ST_LRFeature value;
   c) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY and CHARACTER VARYING feature id values.
   d) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY, CHARACTER VARYING feature id and *ST_Referent* ARRAY values.
2) ST_FeatureID: observes and mutates the feature id value of the ST_LRFeature value.
3) ST_Referents: observes and mutates the referent collection value of the ST_LRFeature value.

4.7.3.2 Functions on ST_LRFeature

1) ST_LRFeatFromText: returns an ST_LRFeature value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRFeature value.
2) ST_LRFeatFromGML: returns an ST_LRFeature value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRFeature value.

4.7.4 ST_LRCurve

The ST_LRCurve subtype of ST_LinearElement specifies any one-dimensional geometry of type ST_Curve which can be linearly measured, that is, which supports the methods of ST_LinearElement. The ST_LRCurve type is instantiable.

4.7.4.1 Methods on ST_LRCurve

1) ST_LRCurve: returns a specified ST_LRCurve value from either:
   a) the well-known text representation of an ST_LRCurve value;
   b) the GML representation of an ST_LRCurve value;
   c) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY and ST_Curve value.
2) ST_Curve: observes and mutates the curve geometry value of the ST_LRCurve value.
3) ST_Point: returns an ST_Point value representing the spatial position spatially equal to the linearly referenced location specified by an ST_PositionExp having an ST_LRCurve subtype of ST_LinearElement.
4) ST_LRPosition: determines the linearly referenced location of a point on the ST_LinearElement of type ST_LRCurve closest to the given ST_Point value using the default Linear Referencing Method of the ST_LRCurve.

4.7.4.2 Functions on ST_LRCurve

1) ST_LRCurveFromText: returns an ST_LRCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRCurve value.
2) ST_LRCurveFromGML: returns an ST_LRCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRCurve value.

4.7.5 ST_LRDirectedEdge

The ST_LRDirectedEdge subtype of ST_LinearElement specifies any one-dimensional topology of type ST_Edge or ST_Link which can be linearly measured, that is, which supports the methods of ST_LinearElement. The ST_LRDirectedEdge type is instantiable.

4.7.5.1 Methods on ST_LRDirectedEdge

1) ST_LRDirectedEdge: returns a specified ST_LRDirectedEdge value from either:
   a) the well-known text representation of an ST_LRDirectedEdge value;
   b) the GML representation of an ST_LRDirectedEdge value;
   c) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY, 'E' (for edge), CHARACTER VARYING topology-name and INTEGER edge ID values.
   d) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY, 'L' (for link), CHARACTER VARYING network-name and INTEGER link ID values.
2) ST_TopologyType: observes and mutates the topology type value of the ST_LRDirectedEdge value.
3) ST_TopoOrNetName: observes and mutates the topology or network name value of the ST_LRDirectedEdge value.
4) ST_EdgeOrLinkID: observes and mutates the edge ID or link ID value of the ST_LRDirectedEdge value.

4.7.5.2 Functions on ST_LRDirectedEdge

1) ST_LREdgeFromText: returns an ST_LRDirectedEdge value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRDirectedEdge value.
2) ST_LREdgeFromGML: returns an ST_LRDirectedEdge value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRDirectedEdge value.

4.7.6 ST_StartValue

The ST_StartValue type specifies an LRM lrmid and its start measure value for a particular ST_LinearElement. The ST_StartValue type is instantiable.

4.7.6.1 Methods on ST_StartValue

1) ST_StartValue: returns a specified ST_StartValue value from the specified LRM lrmid and measure values.
2) ST_LRM: observes and mutates the LRM lrmid value of the ST_StartValue value.
3) ST_Measure: observes and mutates the measure value of the ST_StartValue value.

4.7.7 ST_LRM

The ST_LRM type specifies the Linear Referencing Method which describes the manner in which measurements are made along (and optionally offset from) a linear element. The types of LRM include absolute, relative, interpolative and local interpolative. The ST_LRM type is instantiable.

4.7.7.1 Methods on ST_LRM

1) ST_LRM: returns a specified ST_LRM value from either:

a) the well-known text representation of an ST_LRM value;

b) the GML representation of an ST_LRM value;

c) the specified INTEGER lrmid, CHARACTER VARYING LRM name, CHARACTER VARYING LRM type, CHARACTER VARYING unit of measure and CHARACTER VARYING ARRAY constraint collection values.

d) the specified INTEGER lrmid, CHARACTER VARYING LRM name, CHARACTER VARYING LRM type, CHARACTER VARYING unit of measure, CHARACTER VARYING ARRAY constraint collection, CHARACTER VARYING offset unit of measure, CHARACTER VARYING positive lateral offset direction and CHARACTER VARYING positive vertical offset direction values.

2) ST_LRMID: observes and mutates the lrmid value of the ST_LRM value.

3) ST_LRMName: observes and mutates the name value of the ST_LRM value.

4) ST_LRMType: observes and mutates the type value of the ST_LRM value.

5) ST_UnitOfMeasure: observes and mutates the units value of the ST_LRM value.

6) ST_Constraints: observes and mutates the constraint collection value of the ST_LRM value.

7) ST_OffsetMeasUnit: observes and mutates the offset units value of the ST_LRM value.

8) ST_PosLatOffsetDir: observes and mutates the positive lateral offset direction value of the ST_LRM value.

9) ST_PosVerOffsetDir: observes and mutates the positive vertical offset direction value of the ST_LRM value.

4.7.7.2 Functions on ST_LRM

1) ST_LRMFromText: returns an ST_LRM value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRM value.

2) ST_LRMFromGML: returns an ST_LRM value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRM value.

4.7.8 ST_DistanceExp

The ST_DistanceExp type specifies the linear referenced measure value. The ST_DistanceExp type is instantiable.

4.7.8.1 Methods on ST_DistanceExp

1) ST_DistanceExp: returns a specified ST_DistanceExp value from either:

a) the well-known text representation of an ST_DistanceExp value;

b) the GML representation of an ST_DistanceExp value;

c) the specified ST_LRMeasure distance along value.

d) the specified ST_LRMeasure distance along and ST_LatOffsetExp lateral offset expression values.

e) the specified ST_LRMeasure distance along and ST_VerOffsetExp vertical offset expression values.

f) the specified ST_LRMeasure distance along, ST_LatOffsetExp lateral offset expression and ST_VerOffsetExp vertical offset expression values.

g) the specified ST_LRMeasure distance along and ST_VectorOffsetExp vector offset expression values.

h) the specified ST_LRMeasure distance along and CHARACTER VARYING "from" referent feature ID and "from" referent name values.

i) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID and "from" referent name and ST_LatOffsetExp lateral offset expression values.

j) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID and "from" referent name and ST_VerOffsetExp vertical offset expression values.

k) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID and "from" referent name, ST_LatOffsetExp lateral offset expression and ST_VerOffsetExp vertical offset expression values.

l) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID and "from" referent name and ST_VectorOffsetExp vector offset expression values.

m) the specified ST_LRMeasure distance along, and CHARACTER VARYING "from" referent feature ID, "from" referent name, "towards" referent feature ID and "towards" referent name values.

n) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, "from" referent name, "towards" referent feature ID and "towards" referent name and ST_LatOffsetExp lateral offset expression values.

o) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, "from" referent name, "towards" referent feature ID and "towards" referent name and ST_VerOffsetExp vertical offset expression values.

p) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, "from" referent name, "towards" referent feature ID and "towards" referent name, ST_LatOffsetExp lateral offset expression and ST_VerOffsetExp vertical offset expression values.

q) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, "from" referent name, "towards" referent feature ID and "towards" referent name and ST_VectorOffsetExp vector offset expression values.

2) ST_DistanceAlong observes and mutates the distance along value of the ST_DistanceExp value.

3) ST_FromRefFeaID: observes and mutates the "from" referent feature ID value of the ST_DistanceExp value.

4) ST_FromRefName: observes and mutates the "from" referent name value of the ST_DistanceExp value.

5) ST_TowardsRefFeaID: observes and mutates the "towards" referent feature ID value of the ST_DistanceExp value.

6) ST_TowardsRefName: observes and mutates the "towards" referent name value of the ST_DistanceExp value.

7) ST_LatOffsetExp: observes and mutates the lateral offset expression value of the ST_DistanceExp value.

8) ST_VerOffsetExp: observes and mutates the vertical offset expression value of the ST_DistanceExp value.

9) ST_VectorOffsetExp: observes and mutates the vector offset expression value of the ST_DistanceExp value.

4.7.8.2 Functions on ST_DistanceExp

1) ST_DisExpFromText: returns an ST_DistanceExp value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_DistanceExp value.

2) ST_DisExpFromGML: returns an ST_DistanceExp value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_DistanceExp value.

4.7.9 ST_LRMeasure

The ST_LRMeasure type specifies a measured value with optional units of measure. The ST_LRMeasure type is instantiable.

4.7.9.1 Methods on ST_LRMeasure

1) ST_LRMeasure: returns a specified ST_LRMeasure value from either:
    a) the specified DOUBLE PRECISION measure value.
    b) the specified DOUBLE PRECISION measure and the CHARACTER VARYING units values.
2) ST_Measure: observes and mutates the measure value of the ST_LRMeasure value.
3) ST_UnitOfMeasure: observes and mutates the units value of the ST_LRMeasure value.

4.7.10 ST_Referent

The ST_Referent type specifies a known location along an owning ST_LRFeature. The ST_Referent type is instantiable.

4.7.10.1 Methods on ST_Referent

1) ST_Referent: returns a specified ST_Referent value from either:
    a) the specified CHARACTER VARYING name and CHARACTER VARYING type values.
    b) the specified CHARACTER VARYING name, CHARACTER VARYING type and ST_Point position values.
    c) the specified CHARACTER VARYING name, CHARACTER VARYING type and ST_PositionExp location values.
    d) the specified ST_LRFeature owner, CHARACTER VARYING name, CHARACTER VARYING type, ST_Point position and ST_PositionExp location values.
2) ST_ReferentName: observes and mutates the referent name value of the ST_Referent value.
3) ST_ReferentType: observes and mutates the referent type value of the ST_Referent value.
4) ST_Position: observes and mutates the position value of the ST_Referent value.
5) ST_Location: observes and mutates the location value of the ST_Referent value.
6) ST_ChangePosAndLoc: mutates both the position and location values of the ST_Referent value.

4.7.11 ST_LatOffsetExp

The ST_LatOffsetExp type specifies the lateral offset for a linearly referenced location. The ST_LatOffsetExp type is instantiable.

4.7.11.1 Methods on ST_LatOffsetExp

1) ST_LatOffsetExp: returns a specified ST_LatOffsetExp value from either:
    a) the specified ST_LRMeasure offset lateral distance value.
    b) the specified ST_LRMeasure offset lateral distance and ST_Geometry lateral offset referent feature geometry values.
    c) the specified ST_LRMeasure offset lateral distance and CHARACTER VARYING lateral offset referent description values.
2) ST_OffsetLatDist: observes and mutates the offset lateral distance value of the ST_LatOffsetExp value.
3) ST_FeatureGeometry: observes and mutates the lateral offset referent feature geometry value of the ST_LatOffsetExp value.
4) ST_OffsetRefDesc: observes and mutates the lateral offset referent descriptoin value of the ST_LatOffsetExp value.

4.7.12 ST_VerOffsetExp

The ST_VerOffsetExp type specifies the vertical offset for a linearly referenced location. The ST_VerOffsetExp type is instantiable.

4.7.12.1 Methods on ST_VerOffsetExp

1) ST_VerOffsetExp: returns a specified ST_VerOffsetExp value from either:
   a) the specified ST_LRMeasure offset vertical distance value.
   b) the specified ST_LRMeasure offset vertical distance and ST_Geometry vertical offset referent feature geometry values.
   c) the specified ST_LRMeasure offset vertical distance and CHARACTER VARYING vertical offset referent description values.
2) ST_OffsetVerDist: observes and mutates the offset vertical distance value of the ST_VerOffsetExp value.
3) ST_FeatureGeometry: observes and mutates the vertical offset referent feature geometry value of the ST_VerOffsetExp value.
4) ST_OffsetRefDesc: observes and mutates the vertical offset referent description value of the ST_VerOffsetExp value.

4.7.13 ST_VectorOffsetExp

The ST_VectorOffsetExp type specifies the vector offset for a linearly referenced location. The ST_VectorOffsetExp type is instantiable.

4.7.13.1 Methods on ST_VectorOffsetExp

1) ST_VectorOffsetExp: returns a specified ST_VectorOffsetExp value from:
   a) the specified ST_Vector ARRAY collection of offset vector values.
2) ST_Vector: observes and mutates the offset vector collection of the ST_VectorOffsetExp value.

4.8 Angle and Direction Types

The following types are supported: ST_Angle and ST_Direction.

ST_Angle and ST_Direction are instantiable and have explicitly defined constructor functions.

Either of these types can be used as the type of a column. Declaring a column to be of a particular type implies that any value of the type or any of its subtypes can be used.

4.8.1 ST_Angle

The ST_Angle type is used to measure the degree of separation of two intersecting lines. The ST_Angle type is instantiable. The rotation (clockwise or counterclockwise) of the angle value represented by the ST_Angle type is not specified in order to maximize the applicability of this type across all disciplines.

4.8.1.1 Methods on ST_Angle

1) ST_Angle: returns a specified ST_Angle value from either:
   a) radians, gradians, or degrees;
   b) degrees and minutes;
   c) degrees, minutes, and seconds;
   d) three points, represented as ST_Point values, such that the angle is the lesser of the two possible rotations measured between the direction from the first point to the second point and the direction from the first point to the third point;
   e) two directions, represented as ST_Direction values, where the angle is the lesser of the two possible rotations measured between the two directions;
   f) two lines, represented as ST_LineString values, such that the angle specified is the lesser of the two possible rotations measured between the respective directions of the two lines, where the direction of a line is the direction from its start point to its end point;
   g) a well-known text representation;

h) a GML representation.

2) ST_Radians: observes and mutates the radians attribute of an ST_Angle value.

3) ST_Degrees: observes and mutates the radians attribute of an ST_Angle value using decimal degrees.

4) ST_DegreeComponent: returns the integer value that represents the degrees part of the degrees, minutes, and seconds representation of the ST_Angle value.

5) ST_MinuteComponent: returns the integer value that represents the minutes part of the degrees, minutes, and seconds representation of the ST_Angle value.

6) ST_SecondComponent: returns the double precision value that represents the seconds part of the degrees, minutes, and seconds representation of the ST_Angle value.

7) ST_String: observes and mutates the radians attribute of an ST_Angle using a space separated string of degrees, minutes, and seconds.

8) ST_Gradians: observes and mutates the radians attribute of an ST_Angle value using gradians.

9) ST_Add: adds the value of an angle to the ST_Angle value.

10) ST_Subtract: subtracts the value of an angle from the ST_Angle value.

11) ST_Multiply: multiplies the ST_Angle value by a numeric value.

12) ST_Divide: divides the ST_Angle value by a non-zero, numeric value.

13) ST_AsText: returns the well-known text representation of an ST_Angle value.

14) ST_GMLToSQL: returns the ST_Angle value for the specified GML representation.

15) ST_AsGML: returns the GML representation of an ST_Angle value.

4.8.1.2 Functions on ST_Angle

1) ST_AngleFromText: returns an ST_Angle value, which is transformed from a CHARACTER VARYING value that represents the well-known text representation of an ST_Angle.

2) ST_AngleFromGML: returns an ST_Angle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Angle.

4.8.1.3 Ordering on ST_Angle

1) ST_OrderingCompare: defines full ordering for the ST_Angle type.

4.8.1.4 SQL Transforms on ST_Angle

1) ST_WellKnownText: is the SQL Transform group that transforms an ST_Angle value to and from a well-known text representation in a CHARACTER VARYING value.

2) ST_WellKnownBinary: is the SQL Transform group that transforms an ST_Angle value to and from a well-known binary representation in a DOUBLE PRECISION value.

3) ST_GML: is the SQL Transform group that transforms an ST_Angle value to and from a GML representation in a CHARACTER LARGE OBJECT value.

4.8.2 ST_Direction

The ST_Direction type is used to express direction, expressible either as an azimuth or bearing. The ST_Direction type is instantiable.

4.8.2.1 Methods on ST_Direction

1) ST_Direction: returns a specified ST_Direction value from either:

   a) radians;

   b) 'N' (for North) or 'S' (for South), an angle, and 'E' (for East) or 'W' (for West);

   c) 'N' (for North) or 'S' (for South) and an angle;

   d) two points, represented as ST_Point values, such that the direction defined is the direction from the first point towards the second point;
   e) a line, represented as an ST_LineString value, such that the direction defined is the direction from the start point of the line to the end point of the line;
   f) a well-known text representation;
   g) a GML representation.
2) ST_Radians: returns the ST_Direction value as a DOUBLE PRECISION value in radians, representing clockwise rotation from True North.
3) ST_AngleNAzimuth: observes and mutates the ST_PrivateAngleNAzimuth attribute of an ST_Direction value.
4) ST_RadianBearing: observes the ST_Direction value represented as a bearing with its angle part expressed in radians.
5) ST_DegreesBearing: returns the ST_Direction value represented as a bearing with its angle part expressed in decimal degrees.
6) ST_DMSBearing: returns the ST_Direction value represented as a bearing with its angle part expressed in degrees, minutes, and seconds.
7) ST_RadianNAzimuth: returns the ST_Direction value represented as a North azimuth with its angle part expressed in radians.
8) ST_DegreesNAzimuth: returns the ST_Direction value represented as a North azimuth with its angle part expressed in decimal degrees.
9) ST_DMSNAzimuth: returns the ST_Direction value represented as a North azimuth with its angle part expressed in degrees, minutes, and seconds.
10) ST_RadianSAzimuth: returns the ST_Direction value represented as a South azimuth with its angle part expressed in radians.
11) ST_DegreesSAzimuth: returns the ST_Direction value represented as a South azimuth with its angle part expressed in decimal degrees.
12) ST_DMSSAzimuth: returns the ST_Direction value represented as a South azimuth with its angle part expressed in degrees, minutes, and seconds.
13) ST_AddAngle: mutates the ST_Direction value by adding an angle.
14) ST_SubtractAngle: mutates the ST_Direction value by subtracting an angle.
15) ST_AsText: returns the well-known text representation of an ST_Direction value.
16) ST_GMLToSQL: returns the ST_Direction value for the specified GML representation.
17) ST_AsGML: returns the GML representation of an ST_Direction value.

4.8.2.2 Functions on ST_Direction

1) ST_DirectionFrmTxt: returns an ST_Direction value, which is transformed from a CHARACTER VARYING value that represents the well-known text representation of an ST_Direction .
2) ST_DirectionFrmGML: returns an ST_Direction value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Direction.

4.8.2.3 Ordering on ST_Direction

1) ST_OrderingCompare: defines full ordering for the ST_Direction type.

4.8.2.4 SQL Transforms on ST_Direction

1) ST_WellKnownText: is the SQL Transform group that transforms an ST_Direction value to and from a well-known text representation in a CHARACTER VARYING value.
2) ST_WellKnownBinary: is the SQL Transform group that transforms an ST_Direction value to and from a well-known binary representation in a DOUBLE PRECISION value.

3) ST_GML: is the SQL Transform group that transforms an ST_Direction value to and from a GML representation in a CHARACTER LARGE OBJECT value.

4.9 Support Types

The following support types are defined: ST_TINElement.

ST_TINElement is instantiable and has constructor functions.

4.9.1 ST_TINElement

The ST_TINElement type is used to specify the information used to construct an ST_TIN surface. Element types include random points, group spot, boundary, breakline, soft break, control contour, break void, drape void, void, hole, stop line and user defined element types.

4.9.1.1 Methods on ST_TINElement

1) ST_TINElement: returns an ST_TINElement value constructed from:
   a) the specified element type, element ID, element tag and element geometry values.
2) ST_ElementType: observes and mutates the element type of an ST_TINElement value.
3) ST_ElementID: observes and mutates the element ID of an ST_TINElement value.
4) ST_ElementTag: observes and mutates the element tag of an ST_TINElement value.
5) ST_ElementGeometry: observes and mutates the element geometry of an ST_TINElement value.
6) ST_IsEmpty: tests if an ST_TINElement value corresponds to the empty set.

4.9.2 ST_Vector

The ST_Vector type is an SQL/MM support type. The ST_Vector type is instantiable. A vector is an ordered set of numbers called coordinates that represent a position in a coordinate system. The coordinates may be in a space of any number of dimensions. It is represented by the following attributes:

x coordinate – the first DOUBLE PRECISION coordinate value

y coordinate – the second DOUBLE PRECISION coordinate value

z coordinate – the third DOUBLE PRECISION coordinate value

4.9.2.1 Methods on ST_Vector

1) ST_Vector: returns an ST_Vector value constructed from either:
   a) the well-known text representation of an ST_Vector value;
   b) the well-known binary representation of an ST_Vector value;
   c) the GML representation of an ST_Vector value;
   d) the specified DOUBLE PRECISION values.
2) ST_X: observes and mutates the first DOUBLE PRECISION coordinate value of an ST_Vector value.
3) ST_Y: observes and mutates the second DOUBLE PRECISION coordinate value of an ST_Vector value.
4) ST_Z: observes and mutates the third DOUBLE PRECISION coordinate value of an ST_Vector value.
5) ST_Coordinates: returns all of the coordinate values as a DOUBLE PRECISION ARRAY value.
6) ST_Is3D: returns true if the ST_Vector value contains three coordinate values.
7) ST_SRID: observes and mutates the spatial reference system identifier of an ST_Vector value.
8) ST_IsEmpty: tests if an ST_Vector value corresponds to the empty set.
9) ST_Equals: tests if an ST_Vector value is equal to another ST_Vector value.
10) ST_WKTToSQL: returns the ST_Vector value for the specified well-known text representation.
11) ST_AsText: returns the well-known text representation for the specified ST_Vector value.

12) ST_WKBToSQL: returns the ST_Vector value for the specified well-known binary representation.

13) ST_AsBinary: returns the well-known binary representation for the specified ST_Vector value.

14) ST_GMLToSQL: returns the ST_Vector value for the specified GML representation.

15) ST_AsGML: returns the GML representation for the specified ST_Vector value.

4.9.2.2 Functions on ST_Vector

1) ST_VectorFromText: returns an ST_Vector value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Vector value.

2) ST_VectorFromWKB: returns an ST_Vector value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Vector value.

3) ST_VectorFromGML: returns an ST_Vector value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Vector value.

4.9.2.3 Ordering on ST_Vector

1) ST_OrderingEquals: is the equals only ordering for the ST_Vector type.

4.9.2.4 SQL Transforms on ST_Vector

1) ST_WellKnownText: is the SQL Transform group that transforms an ST_Vector value to and from a well-known text representation in a CHARACTER LARGE OBJECT value.

2) ST_WellKnownBinary: is the SQL Transform group that transforms an ST_Vector value to and from a well-known binary representation in a DOUBLE PRECISION value.

3) ST_GML: is the SQL Transform group that transforms an ST_Vector value to and from a GML representation in a CHARACTER LARGE OBJECT value.

4.9.3 ST_AffinePlacement

The ST_AffinePlacement type is an SQL/MM support type. The ST_AffinePlacement type is instantiable. ST_AffinePlacement defines a linear transformation from a constructive parameter space to the coordinate space of the coordinate reference system being used. It is represented by the following attributes:

location – the ST_Point value which is the target of the parameter space origin

reference directions – the ARRAY of ST_Vector values which are the target directions for the coordinate basis vectors of the parameter space which is the target of the parameter space origin.

4.9.3.1 Methods on ST_AffinePlacement

1) ST_AffinePlacement: returns an ST_AffinePlacement value constructed from the specified ST_Point and ST_Vector ARRAY values.

2) ST_Location: observes and mutates the ST_Point location value in the ST_AffinePlacement value which is the target of the parameter space origin.

3) ST_RefDirections: observes and mutates the collection of ST_Vector reference direction values of an ST_AffinePlacement value which is the target of the parameter space origin.

4) ST_InDimension: returns the INTEGER in dimension value of an ST_AffinePlacement value as the dimension of the input parameter space which is equal to the number of reference directions.

5) ST_OutDimension: returns the INTEGER out dimension value of an ST_AffinePlacement value as the dimension of the output parameter space which is equal to the number of reference directions.

6) ST_Transform: maps a parameter coordinate point to the corresponding coordinate point in the output Cartesian space.

7) ST_IsEmpty: tests if an ST_AffinePlacement value corresponds to the empty set.

4.9.4 ST_NURBSPoint

The ST_NURBSPoint type is an SQL/MM support type. The ST_NURBSPoint type is instantiable. An ST_NURBSPoint is a control point which has been adjusted to consider its respective weight value. The ST_NURBSPoint is represented by the following attributes:

weighted point – the weighted ST_Point value whose coordinate values include consideration of the weight value

weight – optional DOUBLE PRECISION divisor for the rational spline control point. For rational curves, all control points must have weight values.

4.9.4.1 Methods on ST_NURBSPoint

1) ST_NURBSPoint: returns an ST_NURBSPoint value constructed from the specified ST_Point and DOUBLE PRECISION values.
2) ST_WeightedPoint: observes and mutates the ST_Point weighted control point value of an ST_NURBSPoint value.
3) ST_Weight: observes and mutates the DOUBLE PRECISION weight value of an ST_NURBSPoint value.
4) ST_IsEmpty: tests if an ST_NURBSPoint value corresponds to the empty set.

4.9.5 ST_Knot

The ST_Knot type is an SQL/MM support type. The ST_Knot type is instantiable. The collection of knots for an ST_NURBSCurve is the strictly increasing DOUBLE PRECISION breakpoints in the independent variable for the curve. Each ST_Knot value represents a knot value and the number of times that value occurs (multiplicity) in the ST_NURBSCurve knot sequence. ST_Knot is represented by the following attributes:

value – the DOUBLE PRECISION value of the knot

multiplicity – the INTEGER number of times which the knot value occurs for the specified curve

4.9.5.1 Methods on ST_Knot

1) ST_Knot: returns an ST_Knot value constructed from the specified DOUBLE PRECISION and INTEGER values.
2) ST_Value: observes and mutates the DOUBLE PRECISION knot value of an ST_Knot value.
3) ST_Multiplicity: observes and mutates the INTEGER multiplicity value of an ST_Knot.
4) ST_IsEmpty: tests if an ST_Knot value corresponds to the empty set.

4.10 Support Routines

4.10.1 ST_Geometry ARRAY and ST_Vector ARRAY Support Routines

4.10.1.1 Support Functions

1) ST_MaxDimension: returns the maximum geometric dimension value of the elements in an ST_Geometry ARRAY value.
2) ST_CheckSRID: if the elements in the ST_Geometry ARRAY or ST_Vector ARRAY value have mixed spatial reference systems, then raises an exception. Otherwise, the function returns the spatial reference system identifier of the elements of the ST_Geometry ARRAY or ST_Vector ARRAY value.

3) ST_GetCoordDim: checks the consistency of ST_Geometry values in an ST_Geometry ARRAY value with respect to the values returned by the ST_Is3D and ST_IsMeasured methods. If there is an inconsistency, then an exception is raised. Otherwise, all ST_Geometry values contained in the parameters have the same coordinate dimension, ST_Is3D value and ST_IsMeasured value. The ST_GetCoordDim function returns that coordinate dimension. If there are no ST_Geometry values in the parameter list, then the ST_GetCoordDim function returns the default value of 2. For ST_Vector ARRAY values, the functionality is the same except that ST_Vector values do not support m coordinate values. For ST_NURBSPoint ARRAY values, the functionality is the same except that the coordinate dimension is of the ST_NURBSPoint.ST_WeightedPoint ST_Point value and also that ST_NURBSPoints do not support m coordinate values.

4) ST_GetIs3D: returns the value for the ST_Is3D method which is consistent across all the ST_Geometry values in an ST_Geometry ARRAY value.

5) ST_GetIsMeasured: returns the value for the ST_IsMeasured method which is consistent across all the ST_Geometry values in an ST_Geometry ARRAY value.

4.10.1.2 Supporting Procedures

1) ST_CheckNulls: if an ST_Geometry ARRAY or ST_Vector ARRAY value is the null value, or if any of its elements is the null value or the empty set, then an exception is raised.

2) ST_CheckConsecDups: if an ST_Geometry ARRAY value has consecutive duplicate elements, then an exception is raised.

4.10.1.3 Supporting Cast Functions

1) ST_ToPointAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Point valued elements to an ST_Point ARRAY value.

2) ST_ToCurveAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Curve valued elements to an ST_Curve ARRAY value.

3) ST_ToLineStringAry Cast: casts an ST_Geometry ARRAY value that contains only ST_LineString valued elements to an ST_LineString ARRAY value.

4) ST_ToCircularAry Cast: casts an ST_Geometry ARRAY value that contains only ST_CircularString valued elements to an ST_CircularString ARRAY value.

5) ST_ToCircleAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Circle valued elements to an ST_Circle ARRAY value.

6) ST_ToGeodesicAry Cast: casts an ST_Geometry ARRAY value that contains only ST_GeodesicString valued elements to an ST_GeodesicString ARRAY value.

7) ST_ToEllipticalAry Cast: casts an ST_Geometry ARRAY value that contains only ST_EllipticalCurve valued elements to an ST_EllipticalCurve ARRAY value.

8) ST_ToNURBSAry Cast: casts an ST_Geometry ARRAY value that contains only ST_NURBSCurve valued elements to an ST_NURBSCurve ARRAY value.

9) ST_ToClothoidAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Clothoid valued elements to an ST_Clothoid ARRAY value.

10) ST_ToSpiralAry Cast: casts an ST_Geometry ARRAY value that contains only ST_SpiralCurve valued elements to an ST_SpiralCurve ARRAY value.

11) ST_ToCompoundAry Cast: casts an ST_Geometry ARRAY value that contains only ST_CompoundCurve valued elements to an ST_CompoundCurve ARRAY value.

12) ST_ToSurfaceAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Surface valued elements to an ST_Surface ARRAY value.

13) ST_ToCurvePolyAry Cast: casts an ST_Geometry ARRAY value that contains only ST_CurvePolygon valued elements to an ST_CurvePolygon ARRAY value.

14) ST_ToPolygonAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Polygon valued elements to an ST_Polygon ARRAY value.

15) ST_ToTriangleAry Cast: casts an ST_Geometry ARRAY value that contains only ST_Triangle valued elements to an ST_Triangle ARRAY value.

16) ST_ToPolyhdrlAry Cast: casts an ST_Geometry ARRAY value that contains only ST_PolyhdrlSurface valued elements to an ST_PolyhdrlSurface ARRAY value.

17) ST_ToTINAry Cast: casts an ST_Geometry ARRAY value that contains only ST_TIN valued elements to an ST_TIN ARRAY value.

18) ST_ToCompSurfAry Cast: casts an ST_Geometry ARRAY value that contains only ST_CompoundSurface valued elements to an ST_CompoundSurface ARRAY value.

19) ST_ToBRepSolidAry Cast: casts an ST_Geometry ARRAY value that contains only ST_BRepSolid valued elements to an ST_BRepSolid ARRAY value.

4.11 Tables with columns using geometry types

A table may be created with one or more columns that have a declared type of ST_Geometry or one of its subtypes. If a specific spatial reference system is associated with a column, then all geometry values in that column shall be in that specific spatial reference system. For base tables, constraints can be used to model this restriction. For derived tables, the derived table shall be defined in such a way as to ensure that all geometry values in the column will be in the spatial reference system that is associated with this column. The following is the general form of a constraint *CR* defined for a column *COL*, with a declared type of ST_Geometry or one of its subtypes, in the base table *TAB* in the schema *SCH* and the catalog *CAT*:

```
ALTER TABLE SCH.TAB
   ADD CONSTRAINT CR CHECK
      ( COL.ST_SRID() = COALESCE (
         ( SELECT SRS_ID
            FROM ST_INFORMTN_SCHEMA.ST_GEOMETRY_COLUMNS
            WHERE
               TABLE_CATALOG = CAT AND
               TABLE_SCHEMA  = SCH AND
               TABLE_NAME    = TAB AND
               COLUMN_NAME   = COL ),
         COL.ST_SRID()
         )
      )
```

4.12 The Spatial Information Schema

This part of ISO/IEC 13249 prescribes an Information Schema called ST_INFORMTN_SCHEMA. It contains views for the following purposes:

- a view ST_GEOMETRY_COLUMNS, which lists the columns whose declared type is ST_Geometry or one of its subtypes;
- a view ST_SPATIAL_REFERENCE_SYSTEMS, which lists the supported spatial reference systems;
- a view ST_UNITS_OF_MEASURE, which lists the supported units of measure;
- a view ST_SIZINGS, which lists implementation-defined meta-variables and their values.

5 Geometry Types

5.1 ST_Geometry Type and Routines

5.1.1 ST_Geometry Type

Purpose

The ST_Geometry type is the maximal supertype of the geometry type hierarchy. All subtypes have position specified in their attributes.

Definition

```
CREATE TYPE ST_Geometry
   AS (
      ST_PrivateDimension SMALLINT DEFAULT -1,
      ST_PrivateCoordinateDimension SMALLINT DEFAULT 2,
      ST_PrivateIs3D SMALLINT DEFAULT 0,
      ST_PrivateIsMeasured SMALLINT DEFAULT 0
   )
   NOT INSTANTIABLE
   NOT FINAL

   METHOD ST_Dimension()
      RETURNS SMALLINT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_CoordDim()
      RETURNS SMALLINT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_GeometryType()
      RETURNS CHARACTER VARYING(ST_MaxTypeNameLength)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_SRID()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_SRID
      (ansrid INTEGER)
      RETURNS ST_Geometry
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Transform
   (ansrid INTEGER)
   RETURNS ST_Geometry
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsSimple()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DIsSimple()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsValid()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Is3D()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsMeasured()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LocateAlong
   (measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_3DLocateAlong
   (measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DLocateBetween
   (start_measure DOUBLE PRECISION,
    end_measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LocateBetween
   (start_measure DOUBLE PRECISION,
    end_measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Boundary()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DBoundary()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Envelope()
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EnvelopeAsPts()
   RETURNS ST_Point ARRAY[2]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MinX()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_MaxX()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

   METHOD ST_MinY()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MaxY()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MinZ()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MaxZ()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MinM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MaxM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ConvexHull()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Buffer
   (adistance DOUBLE PRECISION)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Buffer
   (adistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Intersection
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DIntersection
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Union
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DUnion
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Difference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_3DDifference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SymDifference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DSymDifference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Distance
   (ageometry ST_Geometry)
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Distance
   (ageometry ST_Geometry,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DDistance
   (ageometry ST_Geometry)
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DDistance
   (ageometry ST_Geometry,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Equals
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DEquals
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Relate
   (ageometry ST_Geometry, amatrix CHARACTER(9))
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Disjoint
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DDisjoint
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Intersects
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DIntersects
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Touches
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Crosses
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Within
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Contains
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Overlaps
   (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToPoint()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToLineString()
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToCircular()
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_ToCircle()
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToGeodesic()
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToElliptical()
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToNURBS()
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToClothoid()
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToSpiral()
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToCompound()
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToCurvePoly()
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_ToPolygon()
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

 METHOD ST_ToPolyhdrlSurf()
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToTIN()
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToCompSurface()
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToBRepSolid()
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToGeomColl()
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToMultiPoint()
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToMultiCurve()
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_ToMultiLine()
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToMultiSurface()
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ToMultiPolygon()
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_WKTToSQL
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsText()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxGeometryAsText)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_WKBToSQL
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsBinary()
   RETURNS BINARY LARGE OBJECT(ST_MaxGeometryAsBinary)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
2) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.
4) *ST_MaxTypeNameLength* is the implementation-defined maximum length used for the character string representation of a type name.
5) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
6) The attribute *ST_PrivateDimension* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateDimension*.
7) The attribute *ST_PrivateCoordinateDimension* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateCoordinateDimension*.
8) The attribute *ST_PrivateIs3D* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_Private3D*.
9) The attribute *ST_PrivateIsMeasured* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateIsMeasured*.

Description

1) The *ST_Geometry* type provides for public use:
   a) a method *ST_Dimension()*,
   b) a method *ST_CoordDim()*,
   c) a method *ST_GeometryType()*,
   d) a method *ST_SRID()*,
   e) a method *ST_SRID(INTEGER)*,
   f) a method *ST_Transform(INTEGER)*,
   g) a method *ST_IsEmpty()*,
   h) a method *ST_IsSimple()*,
   i) a method *ST_3DIsSimple()*,
   j) a method *ST_IsValid()*,
   k) a method *ST_Is3D()*,
   l) a method *ST_IsMeasured()*,
   m) a method *ST_LocateAlong(DOUBLE PRECISION)*,
   n) a method *ST_3DLocateAlong(DOUBLE PRECISION)*,
   o) a method *ST_LocateBetween(DOUBLE PRECISION, DOUBLE PRECISION)*,
   p) a method *ST_3DLocateBetween(DOUBLE PRECISION, DOUBLE PRECISION)*,
   q) a method *ST_Boundary()*,
   r) a method *ST_3DBoundary()*,

s) a method *ST_Envelope()*,

t) a method *ST_EnvelopeAsPts()*,

u) a method *ST_MinX(),*

v) a method *ST_MaxX(),*

w) a method *ST_MinY(),*

x) a method *ST_MaxY(),*

y) a method *ST_MinZ(),*

z) a method *ST_MaxZ(),*

aa) a method *ST_MinM(),*

ab) a method *ST_MaxM(),*

ac) a method *ST_ConvexHull()*,

ad) a method *ST_Buffer(DOUBLE PRECISION)*,

ae) a method *ST_Buffer(DOUBLE PRECISION, CHARACTER VARYING)*,

af) a method *ST_Intersection(ST_Geometry)*,

ag) a method *ST_3DIntersection(ST_Geometry)*,

ah) a method *ST_Union(ST_Geometry)*,

ai) a method *ST_3DUnion(ST_Geometry)*,

aj) a method *ST_Difference(ST_Geometry)*,

ak) a method *ST_3DDifference(ST_Geometry)*,

al) a method *ST_SymDifference(ST_Geometry)*,

am) a method *ST_3DSymDifference(ST_Geometry)*,

an) a method *ST_Distance(ST_Geometry)*,

ao) a method *ST_Distance(ST_Geometry, CHARACTER VARYING)*,

ap) a method *ST_3DDistance(ST_Geometry)*,

aq) a method *ST_3DDistance(ST_Geometry, CHARACTER VARYING)*,

ar) a method *ST_Equals(ST_Geometry)*,

as) a method *ST_3DEquals(ST_Geometry)*,

at) a method *ST_Relate(ST_Geometry, CHARACTER)*,

au) a method *ST_Disjoint(ST_Geometry)*,

av) a method *ST_3DDisjoint(ST_Geometry)*,

aw) a method *ST_Intersects(ST_Geometry)*,

ax) a method *ST_3DIntersects(ST_Geometry)*,

ay) a method *ST_Touches(ST_Geometry)*,

az) a method *ST_Crosses(ST_Geometry)*,

ba) a method *ST_Within(ST_Geometry)*,

bb) a method *ST_Contains(ST_Geometry)*,

bc) a method *ST_Overlaps(ST_Geometry)*,

bd) a method *ST_WKTToSQL(CHARACTER LARGE OBJECT)*,

be) a method *ST_AsText()*,

bf) a method *ST_WKBToSQL(BINARY LARGE OBJECT)*,

bg) a method *ST_AsBinary()*,

bh) a method *ST_GMLToSQL(CHARACTER LARGE OBJECT)*,

bi) a method *ST_AsGML()*,

bj) a function *ST_GeomFromText(CHARACTER LARGE OBJECT)*,

bk) a function *ST_GeomFromText(CHARACTER LARGE OBJECT, INTEGER)*,

bl) a function *ST_GeomFromWKB(BINARY LARGE OBJECT)*,

bm) a function *ST_GeomFromWKB(BINARY LARGE OBJECT, INTEGER)*,

bn) a function *ST_GeomFromGML(CHARACTER LARGE OBJECT)*,

bo) a function *ST_GeomFromGML(CHARACTER LARGE OBJECT, INTEGER)*,

bp) an ordering function *ST_OrderingEquals*(*ST_Geometry, ST_Geometry)*,

bq) an SQL Transform group *ST_WellKnownText*,

br) an SQL Transform group *ST_WellKnownBinary*,

bs) an SQL Transform group *ST_GML*,

bt) an implicit cast of an *ST_Geometry* value to an *ST_Point* value,

bu) an implicit cast of an *ST_Geometry* value to an *ST_LineString* value,

bv) an implicit cast of an *ST_Geometry* value to an *ST_CircularString* value,

bw) an implicit cast of an *ST_Geometry* value to an *ST_Circle* value,

bx) an implicit cast of an *ST_Geometry* value to an *ST_GeodesicString* value,

by) an implicit cast of an *ST_Geometry* value to an *ST_EllipticalCurve* value,

bz) an implicit cast of an *ST_Geometry* value to an *ST_NURBSCurve* value,

ca) an implicit cast of an *ST_Geometry* value to an *ST_Clothoid* value,

cb) an implicit cast of an *ST_Geometry* value to an *ST_SpiralCurve* value,

cc) an implicit cast of an *ST_Geometry* value to an *ST_CompoundCurve* value,

cd) an implicit cast of an *ST_Geometry* value to an *ST_CurvePolygon* value,

ce) an implicit cast of an *ST_Geometry* value to an *ST_Polygon* value,

cf) an implicit cast of an *ST_Geometry* value to an *ST_PolyhdrlSurf* value,

cg) an implicit cast of an *ST_Geometry* value to an *ST_TIN* value,

ch) an implicit cast of an *ST_Geometry* value to an *ST_CompoundSurface* value,

ci) an implicit cast of an *ST_Geometry* value to an *ST_BRepSolid* value,

cj) an implicit cast of an *ST_Geometry* value to an *ST_GeomCollection* value,

ck) an implicit cast of an *ST_Geometry* value to an *ST_MultiPoint* value,

cl) an implicit cast of an *ST_Geometry* value to an *ST_MultiCurve* value,

cm) an implicit cast of an *ST_Geometry* value to an *ST_MultiLineString* value,

cn) an implicit cast of an *ST_Geometry* value to an *ST_MultiSurface* value,

co) an implicit cast of an *ST_Geometry* value to an *ST_MultiPolygon* value.

2) The *ST_PrivateDimension* attribute contains the dimension of the *ST_Geometry* value:

   Case:

   a) If the *ST_Geometry* value corresponds to the empty set, then the dimension is -1.

   b) If the *ST_Geometry* value is a 0-dimensional geometry, then the dimension is 0 (zero).

   c) If the *ST_Geometry* value is a 1-dimensional geometry, then the dimension is 1 (one).

d) If the *ST_Geometry* value is a 2-dimensional geometry, then the dimension is 2.

e) If the *ST_Geometry* value is a 3-dimensional geometry, then the dimension is 3.

3) The *ST_PrivateCoordinateDimension* attribute contains the coordinate dimension of the *ST_Geometry* value.

4) The *ST_PrivateCoordinateDimension* attribute shall be:

Case:

a) 2 for *ST_Geometry* values in two-dimensional coordinate space (R^2).

b) 3 for *ST_Geometry* values in three-dimensional coordinate space where:

i) all points in the *ST_Geometry* value have x, y and z coordinate values, or

ii) all points in the *ST_Geometry* value have x, y and m coordinate values.

c) 4 for *ST_Geometry* values in four-dimensional coordinate space where all points in the *ST_Geometry* value have x, y, z, and m coordinate values.

5) Case:

a) If all the points in the *ST_Geometry* value have z coordinate values, then *ST_PrivateIs3D* is 1 (one).

b) Otherwise, the *ST_PrivateIs3D* is 0 (zero).

6) Case:

a) If all the data points in the *ST_Geometry* value have m coordinate values, then *ST_PrivateIsMeasured* is 1 (one).

b) Otherwise, the *ST_PrivateIsMeasured* is 0 (zero).

7) The value of the *ST_PrivateDimension* attribute shall be less than or equal to the value of the *ST_PrivateCoordinateDimension* attribute, where m is not counted as a coordinate dimension.

8) All instantiable *ST_Geometry* subtypes are defined so that simple values of the geometry type are topologically closed.

9) The coordinate dimension shall be the same as the coordinate dimension of the spatial reference system for the *ST_Geometry* value.

10) An *ST_Geometry* value has an associated spatial reference system specified by a spatial reference system identifier.

5.1.2 ST_Dimension Method

Purpose

Return the dimension of the ST_Geometry value.

Definition

```
CREATE METHOD ST_Dimension()
   RETURNS SMALLINT
   FOR ST_Geometry
   RETURN SELF.ST_PrivateDimension
```

Description

1) The method *ST_Dimension()* has no input parameters.
2) The null-call method *ST_Dimension()* returns the value of the *ST_PrivateDimension* attribute.

5.1.3 ST_CoordDim Method

Purpose

Return the coordinate dimension of the ST_Geometry value.

Definition

```
CREATE METHOD ST_CoordDim()
   RETURNS SMALLINT
   FOR ST_Geometry
   RETURN SELF.ST_PrivateCoordinateDimension
```

Description

1) The method *ST_CoordDim()* has no input parameters.
2) The null-call method *ST_CoordDim()* returns the value of the *ST_PrivateCoordinateDimension* attribute.

5.1.4 ST_GeometryType Method

Purpose

Return the geometry type of the ST_Geometry value.

Definition

```
CREATE METHOD ST_GeometryType()
   RETURNS CHARACTER VARYING(ST_MaxTypeNameLength)
   FOR ST_Geometry
   RETURN
      CASE
         WHEN SELF IS OF (ST_Point) THEN
            'ST_Point'
         WHEN SELF IS OF (ST_LineString) THEN
            'ST_LineString'
         WHEN SELF IS OF (ST_CircularString) THEN
            'ST_CircularString'
         WHEN SELF IS OF (ST_Circle) THEN
            'ST_Circle'
         WHEN SELF IS OF (ST_GeodesicString) THEN
            'ST_GeodesicString'
         WHEN SELF IS OF (ST_EllipticalCurve) THEN
            'ST_EllipticalCurve'
         WHEN SELF IS OF (ST_NURBSCurve) THEN
            'ST_NURBSCurve'
         WHEN SELF IS OF (ST_Clothoid) THEN
            'ST_Clothoid'
         WHEN SELF IS OF (ST_SpiralCurve) THEN
            'ST_SpiralCurve'
         WHEN SELF IS OF (ST_CompoundCurve) THEN
            'ST_CompoundCurve'
         WHEN SELF IS OF (ST_Triangle) THEN
            'ST_Triangle'
         WHEN SELF IS OF (ST_Polygon) THEN
            'ST_Polygon'
         WHEN SELF IS OF (ST_CurvePolygon) THEN
            'ST_CurvePolygon'
         WHEN SELF IS OF (ST_TIN) THEN
            'ST_TIN'
         WHEN SELF IS OF (ST_PolyhdrlSurface) THEN
            'ST_PolyhdrlSurface'
         WHEN SELF IS OF (ST_CompoundSurfaceTHEN
            'ST_CompoundSurface'
         WHEN SELF IS OF (ST_BRepSolid) THEN
            'ST_BRepSolid'
         WHEN SELF IS OF (ST_GeomCollection) THEN
            CASE
               WHEN SELF IS OF (ST_MultiPoint) THEN
                  'ST_MultiPoint'
               WHEN SELF IS OF (ST_MultiLineString) THEN
                  'ST_MultiLineString'
               WHEN SELF IS OF (ST_MultiCurve) THEN
                  'ST_MultiCurve'
               WHEN SELF IS OF (ST_MultiPolygon) THEN
                  'ST_MultiPolygon'
               WHEN SELF IS OF (ST_MultiSurface) THEN
                  'ST_MultiSurface'
               ELSE
                  'ST_GeomCollection'
               END
```

```
        -- ELSE
           --
           -- See Description
           --
    END
```

Definitional Rules

1) *ST_MaxTypeNameLength* is the implementation-defined maximum length used for the character string representation of a type name.

Description

1) The method *ST_GeometryType()* has no input parameters.
2) For the null-call method *ST_GeometryType()*:

   Case:

   a) If SELF is of type *ST_Point*, then return the value: 'ST_Point'.
   b) If SELF is of type *ST_LineString*, then return the value: 'ST_LineString'.
   c) If SELF is of type *ST_CircularString*, then return the value 'ST_CircularString'.
   d) If SELF is of type *ST_Circle*, then return the value 'ST_Circle'.
   e) If SELF is of type *ST_GeodesicString*, then return the value 'ST_GeodesicString'.
   f) If SELF is of type *ST_EllipticalCurve*, then return the value 'ST_EllipticalCurve'.
   g) If SELF is of type *ST_NURBSCurve*, then return the value 'ST_NURBSCurve'.
   h) If SELF is of type *ST_Clothoid*, then return the value 'ST_Clothoid'.
   i) If SELF is of type *ST_SpiralCurve*, then return the value 'ST_SpiralCurve'.
   j) If SELF is of type *ST_CompoundCurve*, then return the value 'ST_CompoundCurve'.
   k) If SELF is of type *ST_Triangle*, then return the value: 'ST_Triangle'.
   l) If SELF is of type *ST_Polygon*, then return the value: 'ST_Polygon'.
   m) If SELF is of type *ST_CurvePolygon*, then return the value: 'ST_CurvePolygon'.
   n) If SELF is of type *ST_TIN*, then return the value: 'ST_TIN'.
   o) If SELF is of type *ST_PolyhdrlSurface*, then return the value: 'ST_PolyhdrlSurface'.
   p) If SELF is of type *ST_CompoundSurface*, then return the value: 'ST_CompoundSurface'.
   q) If SELF is of type *ST_BRepSolid*, then return the value: 'ST_BRepSolid'.
   r) If SELF is of type *ST_GeomCollection*, then:

      Case:

      i) If SELF is of type *ST_MultiPoint*, then return the value 'ST_MultiPoint'.
      ii) If SELF is of type *ST_MultiLineString*, then return the value 'ST_MultiLineString'.
      iii) If SELF is of type *ST_MultiCurve*, then return the value 'ST_MultiCurve'.
      iv) If SELF is of type *ST_MultiPolygon*, then return the value 'ST_MultiPolygon'.
      v) If SELF is of type *ST_MultiSurface*, then return the value 'ST_MultiSurface'.
      vi) Otherwise, return the value: 'ST_GeomCollection'.

   s) Otherwise, the method *ST_GeometryType()* returns an implementation-defined CHARACTER VARYING value for a user-defined type not defined in this part of ISO/IEC 13249.

5.1.5 ST_SRID Methods

Purpose

Observe and mutate the spatial reference system identifier of the ST_Geometry value.

Definition

```
CREATE METHOD ST_SRID()
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_SRID
   (ansrid INTEGER)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_SRID()* has no input parameters.
2) The null-call method *ST_SRID()* returns the spatial reference system identifier for the *ST_Geometry* value.
3) The method *ST_SRID(INTEGER)* takes the following input parameters:
   a) an INTEGER value *ansrid*.
4) The parameter *ansrid* is a spatial reference system identifier.
5) The null-call type-preserving method *ST_SRID(INTEGER)* returns an *ST_Geometry* value with the spatial reference system identifier set to *ansrid*.

5.1.6 ST_Transform Method

Purpose

Return an ST_Geometry value transformed to the specified spatial reference system, considering z and m coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Transform
   (ansrid INTEGER)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Transform(INTEGER)* takes the following input parameters:
   a) an INTEGER value *ansrid*.
2) The parameter *ansrid* is a spatial reference system identifier.
3) For the null-call type-preserving method *ST_Transform(INTEGER)*:

   Case:

   a) If the spatial reference system identifier of SELF is equal to *ansrid*, then return SELF.
   b) If SELF is an empty set, then return SELF with the spatial reference system identifier equal to *ansrid*.
   c) If SELF cannot be transformed to the spatial reference system specified by *ansrid*, then an exception condition is raised: *SQL/MM Spatial exception – failed to transform geometry.*
   d) Otherwise, return an *ST_Geometry* value as the result of an implementation-defined transform of SELF from the spatial reference system of SELF to the spatial reference system specified by *ansrid*. The value returned has the spatial reference system identifier equal to *ansrid*.
4) If SELF.ST_SRID() supports z or m coordinate values and the geometry of SELF contains ST_Point values with z or m coordinate values, then the spatial reference system specified by *ansrid* must also support z or m coordinate values or else SELF cannot be transformed to the spatial reference system specified by *ansrid*.
5) If SELF.ST_Is3D() is equal to 1 (one), then:
   a) The z coordinate values are considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value shall include z coordinate values.
6) If SELF.ST_IsMeasured() is equal to 1 (one), then:
   a) The m coordinate values are considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value shall include m coordinate values.

5.1.7 ST_IsEmpty Method

Purpose

Test if an ST_Geometry value corresponds to the empty set.

Definition

```
CREATE METHOD ST_IsEmpty()
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsEmpty()* has no input parameters.

2) For the null-call method *ST_IsEmpty()*:

   Case:

   a) If the *ST_Geometry* value corresponds to the empty set, then return 1 (one).

   b) Otherwise, return 0 (zero).

3) The Description sections in the subclauses defining the subtypes of *ST_Geometry* include the specific conditions that cause a value of these types to correspond to the empty set.

5.1.8 ST_IsSimple Method

Purpose

Test if an ST_Geometry value has no anomalous geometric points, such as self intersection, ignoring z coordinate values in the determination.

Definition

```
CREATE METHOD ST_IsSimple()
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsSimple()* has no input parameters.

2) For the null-call method *ST_IsSimple()*:

   Case:

   a) If SELF is an empty set, then return 1 (one).

   b) If the *ST_Geometry* value is simple, then return 1 (one).

   c) Otherwise, return 0 (zero).

3) The Description sections in the subclauses defining the subtypes of *ST_Geometry* include the specific conditions that cause a value of these types to be classified as simple.

4) If SELF.*ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the determination.

5.1.9 ST_3DIsSimple Method

Purpose

Test if an ST_Geometry value has no anomalous geometric points, such as self intersection, considering z coordinate values in the determination.

Definition

```
CREATE METHOD ST_3DIsSimple()
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DIsSimple()* has no input parameters.
2) For the null-call method *ST_3DIsSimple()*:

   Case:

   a) If SELF is an empty set, then return 1 (one).
   b) If the *ST_Geometry* value is simple, then return 1 (one).
   c) Otherwise, return 0 (zero).
3) The Description sections in the subclauses defining the subtypes of *ST_Geometry* include the specific conditions that cause a value of these types to be classified as simple.
4) If SELF.ST_Is3D() is equal to 1 (one), then the z coordinate values are considered in the determination.

5.1.10 ST_IsValid Method

Purpose

Test if an ST_Geometry value is well formed.

Definition

```
CREATE METHOD ST_IsValid()
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsValid()* has no input parameters.
2) For the null-call method *ST_IsValid()*:

   Case:

   a) If SELF is an empty *set*, then return 1 (one).
   b) If the *ST_Geometry* value is well formed, then return 1 (one).
   c) Otherwise, return 0 (zero).
3) The Description sections in the subclauses defining the subtypes of *ST_Geometry* include the specific conditions that cause a value of these types to be classified as well formed.

5.1.11 ST_Is3D Method

Purpose

Test if an ST_Geometry value has z coordinate values.

Definition

```
CREATE METHOD ST_Is3D()
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN SELF.ST_PrivateIs3D
```

Description

1) The method *ST_Is3D()* has no input parameters.
2) The null-call method *ST_Is3D()* returns the value of the *ST_PrivateIs3D* attribute.

5.1.12 ST_IsMeasured Method

Purpose

Test if an ST_Geometry value has m coordinate values.

Definition

```
CREATE METHOD ST_IsMeasured()
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN SELF.ST_PrivateIsMeasured
```

Description

1) The method *ST_IsMeasured()* has no input parameters.
2) The null-call method *ST_IsMeasured()* returns the value of the *ST_PrivateIsMeasured* attribute.

5.1.13 ST_LocateAlong Method

Purpose

Return a derived geometry value that matches the specified measure, ignoring z coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_LocateAlong
   (measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   FOR ST_Geometry
   RETURN ST_LocateBetween(measure, measure)
```

Description

1) The method *ST_LocateAlong(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *measure*.

2) The null-call method *ST_LocateAlong(DOUBLE PRECISION)* returns the result of the value expression: *ST_LocateBetween(measure, measure)*.

5.1.14 ST_3DLocateAlong Method

Purpose

Return a derived geometry value that matches the specified measure, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DLocateAlong
   (measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   FOR ST_Geometry
   RETURN ST_3DLocateBetween(measure, measure)
```

Description

1) The method *ST_3DLocateAlong(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *measure*.
2) The null-call method *ST_3DLocateAlong(DOUBLE PRECISION)* returns the result of the value expression: *ST_3DLocateBetween(measure, measure)*.

5.1.15 ST_LocateBetween Method

Purpose

Return a derived geometry collection value with elements that match the specified range of measures inclusively, ignoring z coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_LocateBetween
   (start_measure DOUBLE PRECISION,
    end_measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_LocateBetween(DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *start_measure*,
   b) a DOUBLE PRECISION value *end_measure*.
2) For the null-call method *ST_LocateBetween(DOUBLE PRECISION, DOUBLE PRECISION)*:

   Case:
   a) If *end_measure* is less than *start_measure*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   b) If SELF is an empty set, then return the null value.
   c) If *SELF.ST_IsMeasured*() is equal to 0 (zero), then return an empty *ST_Point* value.
   d) If *SELF.ST_Dimension()* is equal to 0 (zero), then

      Case:
      i) If SELF contains any *ST_Point* values with m coordinate values between *start_measure* and *end_measure* inclusively, then return an *ST_MultiPoint* value that contains the *ST_Point* values with m coordinate values between *start_measure* and *end_measure* inclusively.
      ii) Otherwise, return an empty *ST_Point* value.
   e) If *SELF.ST_Dimension()* is equal to 1, then use the implementation-defined interpolation algorithm to estimate values between *start_measure* and *end_measure* inclusively.

      Case:
      i) If the result does not contain any points, then return an empty *ST_Point* value.
      ii) If the *ST_GeomCollection* type is instantiable, then:

         Case:
         1) If the result only contains consecutive points with m coordinate values between *start_measure* and *end_measure* inclusively, then return an *ST_MultiCurve* value with elements of type *ST_Curve* are constructed to represent the curve elements comprised of these consecutive points.
         2) If the result only contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then return an *ST_MultiPoint* value with elements of type *ST_Point* constructed to represent these points.

3) Otherwise:

A) Return an *ST_GeomCollection* value containing:

I) elements of type *ST_Curve* constructed to represent the curve elements comprised of consecutive points with m coordinate values between *start_measure* and *end_measure* inclusively.

II) elements of type *ST_Point* are constructed to represent disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively.

iii) Otherwise:

Case:

1) If the result contains consecutive points with m coordinate values between *start_measure* and *end_measure* inclusively, then:

A) If the result also contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then it is implementation-defined whether or not the following completion condition is raised: *SQL/MM Spatial warning – disconnected points not included in result.*

B) Return an *ST_MultiCurve* value containing elements of type *ST_Curve* constructed to represent the curve elements between these consecutive points.

2) Otherwise, the result contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then an *ST_MultiPoint* value containing elements of type *ST_Point* are constructed to represent these points.

f) If *SELF.ST_Dimension()* is equal to 2, then the operation is implementation-defined.

3) If *SELF.ST_Is3D()* is equal to 1 (one), then he z coordinate values are not considered in the calculation and they are not included in the resultant geometry.

4) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.16 ST_3DLocateBetween Method

Purpose

Return a derived geometry collection value with elements that match the specified range of measures inclusively, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DLocateBetween
   (start_measure DOUBLE PRECISION,
    end_measure DOUBLE PRECISION)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DLocateBetween(DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *start_measure*,

   b) a DOUBLE PRECISION value *end_measure*.

2) For the null-call method *ST_3DLocateBetween(DOUBLE PRECISION, DOUBLE PRECISION)*:

   Case:

   a) If *end_measure* is less than *start_measure*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   b) If SELF is an empty set, then return the null value.

   c) If *SELF.ST_IsMeasured*() is equal to 0 (zero), then return an empty *ST_Point* value.

   d) If *SELF.ST_Dimension()* is equal to 0 (zero), then

      Case:

      i) If SELF contains any *ST_Point* values with m coordinate values between *start_measure* and *end_measure* inclusively, then return an *ST_MultiPoint* value that contains the *ST_Point* values with m coordinate values between *start_measure* and *end_measure* inclusively.

      ii) Otherwise, return an empty *ST_Point* value.

   e) If *SELF.ST_Dimension()* is equal to 1, then use the implementation-defined interpolation algorithm to estimate values between *start_measure* and *end_measure* inclusively.

      Case:

      i) If the result does not contain any points, then return an empty *ST_Point* value.

      ii) If the *ST_GeomCollection* type is instantiable, then:

         Case:

         1) If the result only contains consecutive points with m coordinate values between *start_measure* and *end_measure* inclusively, then return an *ST_MultiCurve* value with elements of type *ST_Curve* are constructed to represent the curve elements comprised of these consecutive points.

         2) If the result only contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then return an *ST_MultiPoint* value with elements of type *ST_Point* constructed to represent these points.

3) Otherwise:

   A) Return an *ST_GeomCollection* value containing:

      I) elements of type *ST_Curve* constructed to represent the curve elements comprised of consecutive points with m coordinate values between *start_measure* and *end_measure* inclusively.

      II) elements of type *ST_Point* are constructed to represent disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively.

iii) Otherwise:

Case:

1) If the result contains consecutive points with m coordinate values between *start_measure* and *end_measure* inclusively, then:

   A) If the result also contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then it is implementation-defined whether or not the following completion condition is raised: *SQL/MM Spatial warning – disconnected points not included in result.*

   B) Return an *ST_MultiCurve* value containing elements of type *ST_Curve* constructed to represent the curve elements between these consecutive points.

2) Otherwise, the result contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then an *ST_MultiPoint* value containing elements of type *ST_Point* are constructed to represent these points.

f) If *SELF.ST_Dimension()* is equal to 2, then the operation is implementation-defined.

3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation and included in the resultant geometry.

4) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.17 ST_Boundary Method

Purpose

Return the boundary of the ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Boundary()
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Boundary()* has no input parameters.
2) For the null-call method *ST_Boundary()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the closure of the boundary of the *ST_Geometry* value: *Closure(Boundary(SELF))*.
3) If SELF.*ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation and they are not included in the resultant geometry.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value include the m coordinate values that were included when SELF was specified.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.18 ST_3DBoundary Method

Purpose

Return the boundary of the ST_Geometry value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DBoundary()
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DBoundary()* has no input parameters.
2) For the null-call method *ST_3DBoundary()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the closure of the boundary of the *ST_Geometry* value: *Closure(Boundary(SELF))*.
3) If *SELF.ST_Is3D()* is equal to 1 (one), then:
   a) The z coordinate values are considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value include the z coordinate values that were included when SELF was specified.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value include the m coordinate values that were included when SELF was specified.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.19 ST_Envelope Method

Purpose

Return the bounding rectangular polygon for the ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Envelope()
   RETURNS ST_Polygon
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Envelope()* has no input parameters.
2) For the null-call method *ST_Envelope()* returns the bounding rectangular polygon for the *ST_Geometry* value:
   a) If SELF is an empty set, then return the null value.
   b) Let *MINX* be the minimum x coordinate value in the *ST_Geometry* value. Let *MINY* be the minimum y coordinate value in the *ST_Geometry* value. Let *MAXX* be the maximum x coordinate value in the *ST_Geometry* value. Let *MAXY* be the maximum y coordinate value in the *ST_Geometry* value.
   c) Let *ETOL* be an implementation-defined envelope tolerance. *ETOL* shall be greater than zero.
   d) If *MINX* is equal to *MAXX*, then set *MINX* to *MINX* - *ETOL* and set *MAXX* to *MAXX* + *ETOL*.
   e) If *MINY* is equal to *MAXY*, then set *MINY* to *MINY* - *ETOL* and set *MAXY* to *MAXY* + *ETOL*.
   f) Return the bounding rectangular polygon constructed as follows:

```
NEW ST_Polygon(
   NEW ST_LineString(
      ARRAY[
         NEW ST_Point(MINX, MINY, SELF.ST_SRID()),
         NEW ST_Point(MAXX, MINY, SELF.ST_SRID()),
         NEW ST_Point(MAXX, MAXY, SELF.ST_SRID()),
         NEW ST_Point(MINX, MAXY, SELF.ST_SRID()),
         NEW ST_Point(MINX, MINY, SELF.ST_SRID())],
      SELF.ST_SRID()),
   SELF.ST_SRID())
```

3) If *SELF.ST_Is3D()* is equal to 1 (one), then:
   a) The z coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Polygon* value do not include z coordinate values.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Polygon* value do not include m coordinate values.
5) The spatial reference system identifier of the returned *ST_Polygon* value is equal to the spatial reference system identifier of SELF.

5.1.20 ST_EnvelopeAsPts Method

Purpose

Return the minimum and maximum coordinate values of an ST_Geometry value as two ST_Point values.

Definition

```
CREATE METHOD ST_EnvelopeAsPts()
   RETURNS ST_Point ARRAY[2]
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_EnvelopeAsPts()* has no input parameters.

2) For the null-call method *ST_EnvelopeAsPts()* returns the minimum and maximum coordinate values of an *ST_Geometry* value as two *ST_Point* values:

   a) If SELF is an empty set, then return the null value.

   b) Let *MINX* be the minimum x coordinate value in the *ST_Geometry* value. Let *MINY* be the minimum y coordinate value in the *ST_Geometry* value. Let *MAXX* be the maximum x coordinate value in the *ST_Geometry* value. Let *MAXY* be the maximum y coordinate value in the *ST_Geometry* value.

   c) If *SELF.Is_3D()* is equal to 1 (one), then let *MINZ* be the minimum z coordinate value in the ST_Geometry value and *MAXZ* the maximum z coordinate value in the ST_Geometry value. Otherwise let *MINZ* = *MAXZ* = NULL.

   d) If *SELF.Is_Measured()* is equal to 1 (one), then let *MINM* be the minimum m coordinate value in the ST_Geometry value and *MAXM* the maximum m coordinate value in the ST_Geometry value. Otherwise let *MINM* = *MAXM* = NULL.

   e) Return the *ST_Point* ARRAY constructed as follows:

```
ARRAY[
   NEW ST_Point(MINX, MINY, MINZ, MINM, SELF.ST_SRID()),
   NEW ST_Point(MAXX, MAXY, MAXZ, MAXM, SELF.ST_SRID())]
```

3) The spatial reference system identifier of the returned *ST_Point* ARRAY value is equal to the spatial reference system identifier of SELF.

5.1.21 ST_MinX Method

Purpose

Return the minimum x coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MinX()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MinX()* has no input parameters.
2) For the null-call method *ST_MinX()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Let *MINX* be the minimum x coordinate value in the *ST_Geometry* value.
      ii) Return the value of *MINX.*

5.1.22 ST_MaxX Method

Purpose

Return the maximum x coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MaxX()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MaxX()* has no input parameters.
2) For the null-call method *ST_MaxX()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Let *MAXX* be the maximum x coordinate value in the *ST_Geometry* value.
      ii) Return the value of *MAXX.*

5.1.23 ST_MinY Method

Purpose

Return the minimum y coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MinY()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MinY()* has no input parameters.
2) For the null-call method *ST_MinY()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Let *MINY* be the minimum y coordinate value in the *ST_Geometry* value.
      ii) Return the value of *MINY.*

5.1.24 ST_MaxY Method

Purpose

Return the maximum y coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MaxY()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MaxY()* has no input parameters.

2) For the null-call method *ST_MaxY()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *MAXY* be the maximum y coordinate value in the *ST_Geometry* value.

      ii) Return the value of *MAXY.*

5.1.25 ST_MinZ Method

Purpose

Return the minimum z coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MinZ()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MinZ()* has no input parameters.
2) For the null-call method *ST_MinZ()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If *SELF.ST_Is3D()* is equal to 0 (zero), then return the null value.
   c) Otherwise:
      i) Let *MINZ* be the minimum z coordinate value in the *ST_Geometry* value.
      ii) Return the value of *MINZ.*

5.1.26 ST_MaxZ Method

Purpose

Return the maximum z coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MaxZ()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MaxZ()* has no input parameters.
2) For the null-call method *ST_MaxZ()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If *SELF.ST_Is3D()* is equal to 0 (zero), then return the null value.
   c) Otherwise:
      i) Let *MAXZ* be the maximum z coordinate value in the *ST_Geometry* value.
      ii) Return the value of *MAXZ.*

5.1.27 ST_MinM Method

Purpose

Return the minimum m coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MinM()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MinM()* has no input parameters.
2) For the null-call method *ST_MinM()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If *SELF.ST_IsMeasured()* is equal to 0 (zero), then return the null value.
   c) Otherwise:
      i) Let *MINM* be the minimum m coordinate value in the *ST_Geometry* value.
      ii) Return the value of *MINM.*

5.1.28 ST_MaxM Method

Purpose

Return the maximum m coordinate value of an ST_Geometry value.

Definition

```
CREATE METHOD ST_MaxM()
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_MaxM()* has no input parameters.

2) For the null-call method *ST_MaxM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If *SELF.ST_IsMeasured()* is equal to 0 (zero), then return the null value.

   c) Otherwise:

      i) Let *MAXM* be the maximum m coordinate value in the *ST_Geometry* value.

      ii) Return the value of *MAXM.*

5.1.29 ST_ConvexHull Method

Purpose

Return the convex hull of the ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_ConvexHull()
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_ConvexHull()* has no input parameters.
2) For the null-call method *ST_ConvexHull()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return an *ST_Geometry* value representing the convex hull of the *ST_Geometry* value.
3) If *SELF.ST_Is3D()* is equal to 1 (one), then:
   a) The z coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.30 ST_Buffer Methods

Purpose

Return the ST_Geometry value that represents all points whose distance from any point of an ST_Geometry value is less than or equal to a specified distance, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Buffer
   (adistance DOUBLE PRECISION)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_Buffer
   (adistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Buffer(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *adistance*.
2) For the null-call method *ST_Buffer(DOUBLE PRECISION)*:
   a) The parameter *adistance* is measured in an implementation-defined linear unit of measure in the spatial reference system of SELF.
   b) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return an *ST_Geometry* value that represents all points whose distance from SELF is less than or equal to *adistance*.
   c) If *SELF.ST_Is3D()* is equal to 1 (one), then:
      i) The z coordinate values are not considered in the calculation.
      ii) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.
   d) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
      i) The m coordinate values are not considered in the calculation.
      ii) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
   e) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

3) The method *ST_Buffer(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:
   a) a DOUBLE PRECISION value *adistance*,
   b) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_Buffer(DOUBLE PRECISION, CHARACTER VARYING)*:
   a) The DOUBLE PRECISION value *adistance* is measured in the units indicated by *aunit*.
   b) The values for *aunit* shall be a supported <unit name>.
   c) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   d) If the unit specified by *aunit* is not supported by the implementation to compute the *ST_Geometry* value that represents all points whose distance from SELF is less than or equal to *adistance*, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
   e) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return an *ST_Geometry* value that represents all points whose distance from SELF is less than or equal to *adistance.*
   f) If *SELF.ST_Is3D()* is equal to 1 (one), then:
      i) The z coordinate values are not considered in the calculation.
      ii) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.
   g) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
      i) The m coordinate values are not considered in the calculation.
      ii) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
   h) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.31 ST_Intersection Method

Purpose

Return an ST_Geometry value that represents the point set intersection of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Intersection
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Intersection(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) The null-call method *ST_Intersection(ST_Geometry)* returns an *ST_Geometry* value that represents the point set intersection: *Closure(SELF* $\cap$ *ageometry)*.

   NOTE For the list of subtypes returned by ST_Intersection(ST_Geometry), see Table 8 — Return Type Matrix for the ST_Intersection Method in Subclause 5.1.39, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".
3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then:
   a) The z coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.32 ST_3DIntersection Method

Purpose

Return an ST_Geometry value that represents the point set intersection of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DIntersection
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DIntersection(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) The null-call method *ST_3DIntersection(ST_Geometry)* returns an *ST_Geometry* value that represents the point set intersection: *Closure(SELF* $\cap$ *ageometry)*.

   NOTE For the list of subtypes returned by ST_3DIntersection(ST_Geometry), see Table 8 — Return Type Matrix for the ST_Intersection Method in Subclause 5.1.22, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".
3) Case:
   a) If *SELF.ST_Is3D()* is equal to 1 (one) and *ageometry.ST_Is3D()* is equal to 1 (one), then:
      i) The z coordinate values are considered in the calculation.
      ii) The *ST_Point* values contained in the returned *ST_Geometry* value include z coordinate values.
   b) Otherwise, an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.33 ST_Union Method

Purpose

Return an ST_Geometry value that represents the point set union of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Union
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Union(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) The null-call method *ST_Union(ST_Geometry)* returns an *ST_Geometry* value that represents the point set union: *Closure(SELF ∪ ageometry)*.

   NOTE For the list of subtypes returned by ST_Union(ST_Geometry), see Table 9 — Return Type Matrix for the ST_Union Method in Subclause 5.1.39, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then:

   a) The z coordinate values are not considered in the calculation.

   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:

   a) The m coordinate values are not considered in the calculation.

   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.

5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.34 ST_3DUnion Method

Purpose

Return an ST_Geometry value that represents the point set union of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DUnion
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DUnion(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) The null-call method *ST_3DUnion(ST_Geometry)* returns an *ST_Geometry* value that represents the point set union: *Closure(SELF ∪ ageometry)*.

   NOTE For the list of subtypes returned by ST_3DUnion(ST_Geometry), see Table 9 — Return Type Matrix for the ST_Union Method in Subclause 5.1.22, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".
3) Case:
   a) If *SELF.ST_Is3D()* is equal to 1 (one) and *ageometry.ST_Is3D()* is equal to 1 (one), then:
      i) The z coordinate values are considered in the calculation.
      ii) The *ST_Point* values contained in the returned *ST_Geometry* value include z coordinate values.
   b) Otherwise, an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.35 ST_Difference Method

Purpose

Return an ST_Geometry value that represents the point set difference of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Difference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Difference(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) The null-call method *ST_Difference(ST_Geometry)* returns an *ST_Geometry* value that represents the point set difference: *Closure(SELF — ageometry)*.

   NOTE For the list of subtypes returned by ST_Difference(ST_Geometry), see Table 10 — Return Type Matrix for the ST_Difference Method in Subclause 5.1.39, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".
3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then:
   a) The z coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are not considered in the calculation.
   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.
5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.36 ST_3DDifference Method

Purpose

Return an ST_Geometry value that represents the point set difference of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DDifference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DDifference(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) The null-call method *ST_3DDifference(ST_Geometry)* returns an *ST_Geometry* value that represents the point set difference: *Closure(SELF — ageometry)*.

   NOTE For the list of subtypes returned by ST_3DDifference(ST_Geometry), see Table 10 — Return Type Matrix for the ST_Difference Method in Subclause 5.1.22, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".

3) Case:

   a) If *SELF.ST_Is3D()* is equal to 1 (one) and *ageometry.ST_Is3D()* is equal to 1 (one), then:

      i) The z coordinate values are considered in the calculation.

      ii) The *ST_Point* values contained in the returned *ST_Geometry* value include z coordinate values.

   b) Otherwise, an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:

   a) The m coordinate values are not considered in the calculation.

   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.

5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.37 ST_SymDifference Method

Purpose

Return an ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_SymDifference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   RETURN SELF.ST_Difference(ageometry).
      ST_Union(ageometry.ST_Difference(SELF))
```

Description

1) The method *ST_SymDifference(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) The null-call method *ST_SymDifference(ST_Geometry)* returns an *ST_Geometry* value that represents the point set symmetric difference: *Closure(Closure(SELF — ageometry) ∪ Closure(ageometry — SELF))*.

   NOTE For the list of subtypes returned by ST_SymDifference(ST_Geometry), see Table 11 — Return Type Matrix for the ST_SymDifference Method in Subclause 5.1.39, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then:

   a) The z coordinate values are not considered in the calculation.

   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include z coordinate values.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:

   a) The m coordinate values are not considered in the calculation.

   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.

5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.38 ST_3DSymDifference Method

Purpose

Return an ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DSymDifference
   (ageometry ST_Geometry)
   RETURNS ST_Geometry
   FOR ST_Geometry
   RETURN SELF.ST_3DDifference(ageometry).
      ST_3DUnion(ageometry.ST_3DDifference(SELF))
```

Description

1) The method *ST_3DSymDifference(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) The null-call method *ST_3DSymDifference(ST_Geometry)* returns an *ST_Geometry* value that represents the point set symmetric difference: *Closure(Closure(SELF — ageometry) ∪ Closure(ageometry — SELF))*.

   NOTE For the list of subtypes returned by ST_3DSymDifference(ST_Geometry), see Table 11 — Return Type Matrix for the ST_SymDifference Method in Subclause 5.1.22, "Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference".

3) Case:

   a) If *SELF.ST_Is3D()* is equal to 1 (one) and *ageometry.ST_Is3D()* is equal to 1 (one), then:

      i) The z coordinate values are considered in the calculation.

      ii) The *ST_Point* values contained in the returned *ST_Geometry* value include z coordinate values.

   b) Otherwise, an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then:

   a) The m coordinate values are not considered in the calculation.

   b) The *ST_Point* values contained in the returned *ST_Geometry* value do not include m coordinate values.

5) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

5.1.39 Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference

The return types from the *ST_Intersection*, *ST_Union*, *ST_Difference*, and *ST_SymDifference* methods on the *ST_Geometry* type are defined in this subclause. These methods take two *ST_Geometry* values, the subject parameter and an additional parameter and return *ST_Geometry* values. The parameter type code for the possible parameter types is described in Table 6 — Parameter Types. For any given method, the type of the return value shall be one of the possible subtypes of *ST_Geometry* as specified in the return type set. The return type set codes are described in Table 7 — Return Type Sets.

A matrix for each method is presented with the parameter type code of the subject parameter down the column and the parameter type code of the additional parameter across the row. Each cell of the matrix contains the return set code for the two parameter types. The matrix for the *ST_Intersection* method is in Table 8 — Return Type Matrix for the ST_Intersection Method. The matrix for the *ST_Union* method is in Table 9 — Return Type Matrix for the ST_Union Method. The matrix for the *ST_Difference* method is in Table 10 — Return Type Matrix for the ST_Difference Method. The matrix for the *ST_SymDifference* method is in Table 11 — Return Type Matrix for the ST_SymDifference Method.

The elements in return values of type *ST_GeomCollection* have no implied order.

Table 6 — Parameter Types

Parameter Type	
Code	Type
∅	empty set of any type
P	*ST_Point*
C	*ST_Curve*
S	*ST_Surface*
D	*ST_Solid*
MP	*ST_MultiPoint*
MC	*ST_MultiCurve*
MS	*ST_MultiSurface*
GC	*ST_GeomCollection*

Table 7 — Return Type Sets

Return Type Sets	
Code	Set of Types
R00	empty set of type *ST_Point*
R01	*ST_Point*
R02	*ST_Curve*
R03	*ST_Surface*
R04	*ST_MultiPoint*,
R05	*ST_MultiCurve*
R06	*ST_MultiSurface*
R07	*ST_GeomCollection*
R08	empty set of type *ST_Point*, *ST_Curve*, *ST_MultiCurve*
R09	empty set of type *ST_Point*, *ST_Point*
R10	empty set of type *ST_Point*, *ST_MultiPoint*
R11	empty set of type *ST_Point*, *ST_Point*, *ST_Curve*, *ST_MultiPoint*, *ST_MultiCurve*, *ST_GeomCollection* of *ST_Point* and *ST_Curve* values
R12	empty set of type *ST_Point*, *ST_Point*, *ST_Curve*, *ST_Surface*, *ST_MultiPoint*, *ST_MultiCurve*, *ST_MultiSurface*, *ST_GeomCollection*
R13	empty set of type *ST_Point*, *ST_Point*, *ST_MultiPoint*
R14	empty set of type *ST_Point*, *ST_Surface*, *ST_MultiSurface*
R15	*ST_Curve*, *ST_GeomCollection* of *ST_Point* and *ST_Curve* values
R16	*ST_Curve*, *ST_MultiCurve*

Return Type Sets	
Code	Set of Types
R17	*ST_MultiCurve*, *ST_GeomCollection* of *ST_Point* and *ST_Curve* values
R18	*ST_MultiSurface*, *ST_GeomCollection* of *ST_Curve* and *ST_Surface* values
R19	*ST_MultiSurface*, *ST_GeomCollection* of *ST_Point* and *ST_Surface* values
R20	*ST_Point*, *ST_MultiPoint*
R21	*ST_Surface*, *ST_GeomCollection* of *ST_Curve* and *ST_Surface* values
R22	*ST_Surface*, *ST_GeomCollection* of *ST_Point* and *ST_Surface* values
R23	*ST_Surface*, *ST_MultiSurface*
R24	*ST_Solid*
R25	*empty set of type ST_Point, ST_Point, ST_Curve, ST_Surface, ST_Solid, ST_MultiPoint, ST_MultiCurve, ST_MultiSurface, ST_GeomCollection*
R26	*ST_Solid, ST_GeomCollection of ST_Point and ST_Solid values*
R27	*ST_Solid, ST_GeomCollection of ST_Curve and ST_Solid values*
R28	*ST_Solid, ST_GeomCollection of ST_Surface and ST_Solid values*
R29	*ST_Solid, ST_GeomCollection of ST_Solid values*
R30	*empty set of type ST_Point, ST_Solid, ST_GeomCollection of ST_Solid values*

Table 8 — Return Type Matrix for the ST_Intersection Method

	a ∩ b								
b a	∅	P	C	S	D	MP	MC	MS	GC
∅	R00	R00	R00	R00	R00	R00	R00	R00	R00
P	R00	R09	R09	R09	R09	R09	R09	R09	R09
C	R00	R09	R11	R11	R11	R13	R11	R11	R11
S	R00	R09	R11	R12	R12	R13	R11	R12	R12
D	R00	R09	R11	R12	R25	R13	R11	R12	R25
MP	R00	R09	R13	R13	R13	R13	R13	R13	R13
MC	R00	R09	R11	R11	R11	R13	R11	R11	R11
MS	R00	R09	R11	R12	R12	R13	R11	R12	R12
GC	R00	R09	R11	R12	R25	R13	R11	R12	R12

Table 9 — Return Type Matrix for the ST_Union Method

	a ∪ b								
b a	∅	P	C	S	D	MP	MC	MS	GC
∅	R00	R01	R02	R03	R24	R04	R05	R06	R07
P	R01	R20	R15	R22	R26	R04	R17	R19	R07
C	R02	R15	R16	R21	R27	R15	R16	R18	R07
S	R03	R22	R21	R23	R28	R22	R21	R23	R07
D	R24	R26	R27	R28	R29	R26	R27	R28	R07
MP	R04	R04	R15	R22	R26	R04	R17	R19	R07
MC	R05	R17	R16	R21	R27	R17	R16	R18	R07
MS	R06	R19	R18	R23	R28	R19	R18	R23	R07
GC	R07	R07	R07	R07	R07	R07	R07	R07	R07

Table 10 — Return Type Matrix for the ST_Difference Method

	a — b								
b \ a	∅	P	C	S	D	MP	MC	MS	GC
∅	R00	R00	R00	R00	R00	R00	R00	R00	R00
P	R01	R09	R09	R09	R09	R09	R09	R09	R09
C	R02	R02	R08	R08	R08	R02	R08	R08	R08
S	R03	R03	R03	R14	R14	R03	R03	R14	R14
D	R24	R24	R24	R24	R30	R24	R24	R24	R30
MP	R04	R13	R13	R13	R13	R13	R13	R13	R13
MC	R05	R05	R08	R08	R08	R05	R08	R08	R08
MS	R06	R06	R06	R14	R14	R06	R06	R14	R14
GC	R07	R12	R12	R12	R25	R12	R12	R12	R12

Table 11 — Return Type Matrix for the ST_SymDifference Method

	(a — b) ∪ (b — a)								
b \ a	∅	P	C	S	D	MP	MC	MS	GC
∅	R00	R01	R02	R03	R24	R04	R05	R06	R07
P	R01	R10	R15	R22	R26	R10	R17	R19	R12
C	R02	R15	R08	R21	R27	R15	R08	R18	R12
S	R03	R22	R21	R14	R28	R22	R21	R14	R12
D	R24	R26	R27	R28	R30	R26	R27	R28	R25
MP	R04	R10	R15	R22	R26	R13	R17	R19	R12
MC	R05	R17	R08	R21	R27	R17	R08	R18	R12
MS	R06	R19	R18	R14	R28	R19	R18	R14	R12
GC	R07	R12	R12	R12	R25	R12	R12	R12	R12

5.1.40 Return Types from ST_3DIntersection, ST_3DUnion, ST_3DDifference, and ST_3DSymDifference

The return types from the *ST_3DIntersection*, *ST_3DUnion*, *ST_3DDifference*, and *ST_3DSymDifference* methods on the *ST_Geometry* type are the same as their 2D counterparts (*ST_Intersection*, *ST_Union*, *ST_Difference*, and *ST_SymDifference*, respectively).

5.1.41 ST_Distance Methods

Purpose

Return the distance between two geometry values, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Distance
   (ageometry ST_Geometry)
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_Distance
   (ageometry ST_Geometry,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Distance(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) For the null-call method *ST_Distance(ST_Geometry)*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *ageometry* is an empty set, then return the null value.
      iii) If SELF and *ageometry* spatially intersect, then return 0 (zero).
      iv) Otherwise, return the distance between two geometries, SELF and *ageometry*, calculated in the spatial reference system of SELF. The distance between two points is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Distance(ST_Geometry)* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_Distance(ST_Geometry)* is in an implementation-defined unit of measure.
3) The method *ST_Distance(ST_Geometry, CHARACTER VARYING)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*;
   b) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_Distance(ST_Geometry, CHARACTER VARYING)*:

   a) The values for *aunit* shall be a supported <unit name>.

   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

   c) If the unit specified by *aunit* is not supported by the implementation to compute the distance between SELF and ageometry, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

   d) Case:

      i) If SELF is an empty set, then return the null value.

      ii) If *ageometry* is an empty set, then return the null value.

      iii) If SELF and *ageometry* spatially intersect, then return 0 (zero).

      iv) Otherwise, return the distance between two geometries, SELF and *ageometry*, calculated in the spatial reference system of SELF. The distance between two points is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.

   e) The returned value is measured in the units indicated by *aunit*.

5.1.42 ST_3DDistance Methods

Purpose

Return the distance between two geometry values, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DDistance
   (ageometry ST_Geometry)
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_3DDistance
   (ageometry ST_Geometry,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DDistance(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) For the null-call method *ST_3DDistance(ST_Geometry)*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *ageometry* is an empty set, then return the null value.
      iii) If *SELF.ST_Is3D()* is equal to 0 (zero) or *ageometry.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.
      iv) If SELF and *ageometry* spatially 3D intersect in 3D, then return 0 (zero).
      v) Otherwise, return the distance between two geometries, SELF and *ageometry*, calculated in the spatial reference system of SELF. The distance between two points is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are considered in the calculation.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DDistance(ST_Geometry)* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_3DDistance(ST_Geometry)* is in an implementation-defined unit of measure.
3) The method *ST_3DDistance(ST_Geometry, CHARACTER VARYING)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*;

b) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_3DDistance(ST_Geometry, CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the distance between SELF and ageometry, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) If *ageometry* is an empty set, then return the null value.

iii) If *SELF.ST_Is3D()* is equal to 0 (zero) or *ageometry.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: SQL/MM *Spatial Exception – both geometries must be 3D*.

iv) If SELF and *ageometry* spatially 3D intersect, then return 0 (zero).

v) Otherwise, return the distance between two geometries, SELF and *ageometry*, calculated in the spatial reference system of SELF. The distance between two points is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are considered in the calculation.

e) The returned value is measured in the units indicated by *aunit*.

5.1.43 ST_Equals Method

Purpose

Test if an ST_Geometry value is spatially 2D equal to another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Equals
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN SELF.ST_SymDifference(ageometry).ST_IsEmpty()
```

Description

1) The method *ST_Equals(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) The null-call method *ST_Equals(ST_Geometry)* returns the result of the value expression: *SELF.ST_SymDifference*(*ageometry*).*ST_IsEmpty()*.

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.44 ST_3DEquals Method

Purpose

Test if an ST_Geometry value is spatially 3D equal to another ST_Geometry value, considering z (but not m) coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DEquals
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DEquals(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) Case:
   a) If *SELF.ST_Is3D()* is equal to 0 (zero) or *ageometry.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.
   b) Otherwise, the null-call method *ST_3DEquals(ST_Geometry)* returns the result of the value expression: *SELF.ST_3DSymDifference(ageometry).ST_IsEmpty()*.
3) Z (but not m) coordinate values are considered in the calculation.

5.1.45 ST_Relate Method

Purpose

Test if an ST_Geometry value is spatially 2D related to another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Relate
   (ageometry ST_Geometry,
    amatrix CHARACTER(9))
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      DECLARE counter INTEGER;
      DECLARE intersectdim INTEGER;

      -- If any value in amatrix is not the list of
      -- possible values: 'T', 'F', '0', '1', '2', '*', then
      -- raise an exception.
      SET counter = 1;
      WHILE counter <= 9 DO
         IF SUBSTRING(amatrix FROM counter FOR 1)
            NOT IN ( 'T', 'F', '0', '1', '2', '*' ) THEN
            SIGNAL SQLSTATE '2FF04'
               SET MESSAGE_TEXT = 'invalid intersection matrix';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      -- Process each of the 9 intersections
      SET counter = 1;
      WHILE counter <= 9 DO
         -- Set intersectdim to the dimension of the current intersection
         CASE counter
            WHEN 1 THEN
               SET intersectdim =
                  Intersection(Interior(SELF), Interior(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 2 THEN
               SET intersectdim =
                  Intersection(Interior(SELF), Boundary(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 3 THEN
               SET intersectdim =
                  Intersection(Interior(SELF), Exterior(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 4 THEN
               SET intersectdim =
                  Intersection(Boundary(SELF), Interior(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 5 THEN
               SET intersectdim =
                  Intersection(Boundary(SELF), Boundary(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 6 THEN
               SET intersectdim =
                  Intersection(Boundary(SELF), Exterior(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 7 THEN
               SET intersectdim =
                  Intersection(Exterior(SELF), Interior(ageometry)).
```

```
                ST_Dimension(); -- See Description
            WHEN 8 THEN
               SET intersectdim =
                  Intersection(Exterior(SELF), Boundary(ageometry)).
                  ST_Dimension(); -- See Description
            WHEN 9 THEN
               SET intersectdim =
                  Intersection(Exterior(SELF), Exterior(ageometry)).
                  ST_Dimension(); -- See Description
         END CASE;
         -- If intersectdim is not in the result set as defined by the
         -- current amatrix position, then return 0 (zero).
         CASE SUBSTRING(amatrix FROM counter FOR 1)
            WHEN 'T' THEN
               IF intersectdim NOT IN ( 0, 1, 2 ) THEN
                  RETURN 0;
               END IF;
            WHEN 'F' THEN
               IF intersectdim <> -1 THEN
                  RETURN 0;
               END IF;
            WHEN '0' THEN
               IF intersectdim <> 0 THEN
                  RETURN 0;
               END IF;
            WHEN '1' THEN
               IF intersectdim <> 1 THEN
                  RETURN 0;
               END IF;
            WHEN '2' THEN
               IF intersectdim <> 2 THEN
                  RETURN 0;
               END IF;
            WHEN '*' THEN
               IF intersectdim NOT IN ( -1, 0, 1, 2 ) THEN
                  RETURN 0;
               END IF;
         END CASE;
         SET counter = counter + 1;
      END WHILE;
      -- If the dimension of each intersection matches the amatrix, then
      -- return 1 (one).
      RETURN 1;
   END
```

Definitional Rules

1) Let *G1* and *G2* be *ST_Geometry* values.
2) *Interior(G1)* represents the interior of *G1*.
3) *Boundary(G1)* represents the boundary of *G1*.
4) *Exterior(G1)* represents the exterior of *G1*.
5) *Intersection(G1, G2)* returns the point set intersection of *G1* and *G2*: $(G1 \cap G2)$

   NOTE interior, boundary, exterior and point set intersection are described in Subclause 4.2.2.1, "The Dimensionally Extended 9 Intersection Model".

Description

1) The method *ST_Relate(ST_Geometry, CHARACTER)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*,

b) a CHARACTER value *amatrix*.

2) For null-call method *ST_Relate(ST_Geometry, CHARACTER)*:

Case:

a) If SELF is an empty set, then return the null value.

b) If *ageometry* is an empty set, then return the null value.

c) If any character in *amatrix* is not 'T', 'F', '0', '1', '2', or '*', then an exception condition is raised: *SQL/MM Spatial exception – invalid intersection matrix*.

d) Otherwise:

i) Each character in *amatrix* corresponds to a cell in the DE9IM pattern matrix. This mapping is defined in Table 12 — DE-9IM .

Table 12 — DE-9IM Mapping

Position	DE-9IM Cell
1	*(Interior(SELF) ∩ Interior(ageometry)).ST_Dimension()*
2	*(Interior(SELF) ∩ Boundary(ageometry)).ST_Dimension()*
3	*(Interior(SELF) ∩ Exterior(ageometry)).ST_Dimension()*
4	*(Boundary(SELF) ∩ Interior(ageometry)).ST_Dimension()*
5	*(Boundary(SELF) ∩ Boundary(ageometry)).ST_Dimension()*
6	*(Boundary(SELF) ∩ Exterior(ageometry)).ST_Dimension()*
7	*(Exterior(SELF) ∩ Interior(ageometry)).ST_Dimension()*
8	*(Exterior(SELF) ∩ Boundary(ageometry)).ST_Dimension()*
9	*(Exterior(SELF) ∩ Exterior(ageometry)).ST_Dimension()*

See Subclause 4.2.2.1, "The Dimensionally Extended 9 Intersection Model" for a detailed description of the DE-9IM.

ii) Each character value in *amatrix* specifies the set of acceptable values for an intersection at a given cell. The meaning for any cell is described in Table 13 — Cell Values.

Table 13 — Cell Values

Cell Value	Intersection Set Results
'T'	{ 0, 1, 2 }
'F'	{ -1 }
'0'	{ 0 }
'1'	{ 1 }
'2'	{ 2 }
'*'	{ -1, 0, 1, 2 }

iii) Let *RESULT* be the value returned by this method. Set *RESULT* to 1 (one).

iv) For *COUNTER* varying from 1 (one) to 9:

1) Let *INTERSECTDIM* be the result of the DE-9IM Intersection at position *COUNTER.*

2) Let *SVI* be the character value at *COUNTER* and let *SRI* be the intersection set results corresponding to *SVI*.

3) If *INTERSECTDIM* is not in the set *SRI*, then set *RESULT* to 0 (zero).

v) Return *RESULT*.

3) If *SELF.ST_Is3D()* is equal to 1 (one) or ageometry.ST_Is3D() is equal to 1 (one), then the z coordinate values are not considered in the calculation.

4) If *SELF.ST_IsMeasured*() is equal to 1 (one) or ageometry.ST_IsMeasured() is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.46 ST_Disjoint Method

Purpose

Test if an ST_Geometry value is spatially 2D disjoint from another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Disjoint
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN SELF.ST_Relate(ageometry, 'FF*FF****')
```

Description

1) The method *ST_Disjoint(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) The null-call method *ST_Disjoint(ST_Geometry)* returns the result of the value expression: *SELF.ST_Relate(ageometry, 'FF*FF****')*.

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.47 ST_3DDisjoint Method

Purpose

Test if an ST_Geometry value is spatially 3D disjoint from another ST_Geometry value, considering z (but not m) coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DDisjoint
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DDisjoint(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) For the null-call method *ST_3DDisjoint(ST_Geometry)*:

   Case:

   a) If *SELF.ST_Is3D()* is equal to 0 (zero) or *ageometry.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.
   b) If SELF and *ageometry* have no points in common (SELF ∩ *ageometry* = ∅), then return 1 (one).
   c) Otherwise, return 0 (zero).
3) Z (but not m) coordinate values are considered in the calculation.

5.1.48 ST_Intersects Method

Purpose

Test if an ST_Geometry value spatially 2D intersects another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Intersects
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN
      CASE SELF.ST_Disjoint(ageometry)
         WHEN 1 THEN
            0
         WHEN 0 THEN
            1
         ELSE
            NULL
      END
```

Description

1) The method *ST_Intersects(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) For the null-call method *ST_Intersects(ST_Geometry)*:

   Case:

   a) If *SELF.ST_Disjoint*(*ageometry*) is equal to 1 (one), then return 0 (zero).

   b) If *SELF.ST_Disjoint*(*ageometry*) is equal to 0 (zero), then return 1 (one).

   c) Otherwise, return the null value.

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.49 ST_3DIntersects Method

Purpose

Test if an ST_Geometry value spatially 3D intersects another ST_Geometry value, considering z (but not m) coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DIntersects
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DIntersects(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) For the null-call method *ST_3DIntersects(ST_Geometry)*:

   Case:

   a) If *SELF.ST_Is3D()* is equal to 0 (zero) or *ageometry.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.
   b) If SELF and *ageometry* have any points in common (SELF ∩ *ageometry* ≠ ∅), then return 1 (one).
   c) Otherwise, return 0 (zero).
3) Z (but not m) coordinate values are considered in the calculation.

5.1.50 ST_Touches Method

Purpose

Test if an ST_Geometry value spatially 2D touches another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Touches
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN
      CASE
         WHEN (SELF.ST_Dimension() = 0 AND
               ageometry.ST_Dimension() = 0) THEN
            NULL
         ELSE
            -- Use ST_Relate to determine result of the touch operation
            -- on SELF and ageometry
            CASE (SELF.ST_Relate(ageometry, 'FT*******') = 1 OR
                  SELF.ST_Relate(ageometry, 'F**T*****') = 1 OR
                  SELF.ST_Relate(ageometry, 'F***T****') = 1)
               WHEN TRUE THEN
                  1
               WHEN FALSE THEN
                  0
               ELSE
                  NULL
            END
      END
```

Description

1) The method *ST_Touches(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) For the null-call method *ST_Touches(ST_Geometry)*:

   Case:

   a) If the dimension of SELF is equal to 0 (zero) and the dimension of *ageometry* is equal to 0 (zero), then return the null value.

   b) Otherwise,

      i) Let *BVE* be the result of the value expression: *SELF.ST_Relate(ageometry, 'FT*******') = 1 OR SELF.ST_Relate(ageometry, 'F**T*****') = 1 OR SELF.ST_Relate(ageometry, 'F***T****') = 1*.

      ii) Case:

         1) If *BVE* is *True*, then return 1 (one).

         2) If *BVE* is *False*, then return 0 (zero).

         3) Otherwise, return the null value.

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.51 ST_Crosses Method

Purpose

Test if an ST_Geometry value spatially 2D crosses another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Crosses
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN
      CASE
         WHEN (SELF.ST_Dimension() = 0 AND
               ageometry.ST_Dimension() = 1) THEN
            SELF.ST_Relate(ageometry, 'T*T******')
         WHEN (SELF.ST_Dimension() = 0 AND
               ageometry.ST_Dimension() = 2) THEN
            SELF.ST_Relate(ageometry, 'T*T******')
         WHEN (SELF.ST_Dimension() = 1 AND
               ageometry.ST_Dimension() = 1) THEN
            SELF.ST_Relate(ageometry, '0********')
         WHEN (SELF.ST_Dimension() = 1 AND
               ageometry.ST_Dimension() = 2) THEN
            SELF.ST_Relate(ageometry, 'T*T******')
         ELSE
            NULL
      END
```

Description

1) The method *ST_Crosses(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) For the null-call method *ST_Crosses(ST_Geometry)*:

   Case:

   a) If the dimension of SELF is equal to 0 (zero) and the dimension of *ageometry* is equal to 1 (one), then return the result of the value expression: *SELF.ST_Relate(ageometry, 'T*T******')*.
   b) If the dimension of SELF is equal to 0 (zero) and the dimension of *ageometry* is equal to 2, then return the result of the value expression: *SELF.ST_Relate(ageometry, 'T*T******')*.
   c) If the dimension of SELF is equal to 1 (one) and the dimension of *ageometry* is equal to 1 (one), then return the result of the value expression: *SELF.ST_Relate(ageometry, '0********')*.
   d) If the dimension of SELF is equal to 1 (one) and the dimension of *ageometry* is equal to 2, then return the result of the value expression: *SELF.ST_Relate(ageometry, 'T*T******')*.
   e) Otherwise, return the null value.
3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.52 ST_Within Method

Purpose

Test if an ST_Geometry value is spatially 2D within another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Within
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN SELF.ST_Relate(ageometry, 'T*F**F***')
```

Description

1) The method *ST_Within(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) The null-call method *ST_Within(ST_Geometry)* returns the result of the value expression: *SELF.ST_Relate(ageometry, 'T*F**F***')*.
3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.53 ST_Contains Method

Purpose

Test if an ST_Geometry value spatially 2D contains another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Contains
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN ageometry.ST_Within(SELF)
```

Description

1) The method *ST_Contains(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
2) The null-call method *ST_Contains(ST_Geometry)* returns the result of the value expression: *ageometry.ST_Within(SELF)*.
3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.54 ST_Overlaps Method

Purpose

Test if an ST_Geometry value spatially 2D overlaps another ST_Geometry value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Overlaps
   (ageometry ST_Geometry)
   RETURNS INTEGER
   FOR ST_Geometry
   RETURN
      CASE
         WHEN (SELF.ST_Dimension() = 0 AND
               ageometry.ST_Dimension() = 0) THEN
            SELF.ST_Relate(ageometry, 'T*T***T**')
         WHEN (SELF.ST_Dimension() = 1 AND
               ageometry.ST_Dimension() = 1) THEN
            SELF.ST_Relate(ageometry, '1*T***T**')
         WHEN (SELF.ST_Dimension() = 2 AND
               ageometry.ST_Dimension() = 2) THEN
            SELF.ST_Relate(ageometry, 'T*T***T**')
         ELSE
            NULL
      END
```

Description

1) The method *ST_Overlaps(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *ageometry*.

2) For the null-call method *ST_Overlaps(ST_Geometry)*:

   Case:

   a) If the dimension of SELF is equal to 0 (zero) and the dimension of *ageometry* is equal to 0 (zero), then return the result of the value expression: *SELF.ST_Relate(ageometry, 'T*T***T**')*.

   b) If the dimension of SELF is equal to 1 (one) and the dimension of *ageometry* is equal to 1 (one), then return the result of the value expression: *SELF.ST_Relate(ageometry, '1*T***T**')*.

   c) If the dimension of SELF is equal to 2 and the dimension of *ageometry* is equal to 2, then return the result of the value expression: *SELF.ST_Relate(ageometry, 'T*T***T**')*.

   d) Otherwise, return the null value.

3) If *SELF.ST_Is3D()* is equal to 1 (one) or *ageometry.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one) or *ageometry.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

5.1.55 Cast

Purpose

Cast an ST_Geometry value to a specific instantiable subtype of ST_Geometry.

Definition

```
CREATE METHOD ST_ToPoint()
   RETURNS ST_Point
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_Point) THEN
         RETURN TREAT(SELF AS ST_Point);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
      IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Point) THEN
               RETURN CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                  AS ST_Point);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_Point().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_Point)
   WITH METHOD ST_ToPoint()
   AS ASSIGNMENT

CREATE METHOD ST_ToLineString()
   RETURNS ST_LineString
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_LineString) THEN
         RETURN TREAT(SELF AS ST_LineString);
      ELSEIF SELF IS OF (ST_CircularString, ST_Circle, ST_GeodesicString,
      ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve,
      ST_CompoundCurve) THEN
         RETURN (SELF AS ST_Curve).ST_CurveToLine();
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Curve) THEN
               RETURN CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                  AS ST_LineString);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_LineString().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_LineString)
   WITH METHOD ST_ToLineString()
   AS ASSIGNMENT
```

```
CREATE METHOD ST_ToCircular()
   RETURNS ST_CircularString
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_CircularString) THEN
         RETURN TREAT(SELF AS ST_CircularString);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_CircularString) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_CircularString);
            END IF;
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_CircularString) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_CircularString);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_CircularString().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_CircularString)
   WITH METHOD ST_ToCircular()
   AS ASSIGNMENT

CREATE METHOD ST_ToCircle()
   RETURNS ST_Circle
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_Circle) THEN
         RETURN TREAT(SELF AS ST_Circle);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_Circle) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_Circle);
            END IF;
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Circle) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_Circle);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
```

```
      END IF;
      RETURN NEW ST_Circle().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_Circle)
   WITH METHOD ST_ToCircle()
   AS ASSIGNMENT

CREATE METHOD ST_ToGeodesic()
   RETURNS ST_GeodesicString
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_GeodesicString) THEN
         RETURN TREAT(SELF AS ST_GeodesicString);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_GeodesicString) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_GeodesicString);
            END IF;
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_GeodesicString) THEN
               RETURN
               CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                  AS ST_GeodesicString);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_GeodesicString().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_GeodesicString)
   WITH METHOD ST_ToGeodesic()
   AS ASSIGNMENT

CREATE METHOD ST_ToElliptical()
   RETURNS ST_EllipticalCurve
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_EllipticalCurve) THEN
         RETURN TREAT(SELF AS ST_EllipticalCurve);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_EllipticalCurve) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_EllipticalCurve);
            END IF;
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_EllipticalCurve) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
```



```
                  AS ST_EllipticalCurve);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_EllipticalCurve().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_EllipticalCurve)
   WITH METHOD ST_ToElliptical()
   AS ASSIGNMENT

CREATE METHOD ST_ToNURBSCurve()
   RETURNS ST_NURBSCurve
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_NURBSCurve) THEN
         RETURN TREAT(SELF AS ST_NURBSCurve);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_NURBSCurve) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_NURBSCurve);
            END IF;
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_NURBSCurve) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_NURBSCurve);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_NURBSCurve().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_NURBSCurve)
   WITH METHOD ST_ToNURBSCurve()
   AS ASSIGNMENT

CREATE METHOD ST_ToClothoid()
   RETURNS ST_Clothoid
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_Clothoid) THEN
         RETURN TREAT(SELF AS ST_Clothoid);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_Clothoid) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_Clothoid);
            END IF;
```

```
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Clothoid) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_Clothoid);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_Clothoid().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_Clothoid)
   WITH METHOD ST_ToClothoid()
   AS ASSIGNMENT

CREATE METHOD ST_ToSpiralCurve()
   RETURNS ST_SpiralCurve
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_SpiralCurve) THEN
         RETURN TREAT(SELF AS ST_SpiralCurve);
      ELSEIF SELF IS OF (ST_CompoundCurve) THEN
         IF (SELF AS ST_CompoundCurve).ST_NumCurves() = 1 THEN
            IF (SELF AS ST_CompoundCurve).ST_CurveN(1)
                  IS OF (ST_SpiralCurve) THEN
               RETURN CAST((SELF AS ST_CompoundCurve).ST_CurveN(1)
                  AS ST_SpiralCurve);
            END IF;
         END IF;
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_SpiralCurve) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_SpiralCurve);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_SpiralCurve().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_SpiralCurve)
   WITH METHOD ST_ToSpiralCurve()
   AS ASSIGNMENT

CREATE METHOD ST_ToCompound()
   RETURNS ST_CompoundCurve
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_CompoundCurve) THEN
         RETURN TREAT(SELF AS ST_CompoundCurve);
```

```
      ELSEIF SELF IS OF (ST_LineString, ST_CircularString, ST_Circle,
      ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid,
      ST_SpiralCurve) THEN
         RETURN NEW ST_CompoundCurve(ARRAY[TREAT(SELF AS ST_Curve)],
            SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Curve) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_CompoundCurve);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_CompoundCurve().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_CompoundCurve)
   WITH METHOD ST_ToCompound()
   AS ASSIGNMENT

CREATE METHOD ST_ToCurvePoly()
   RETURNS ST_CurvePolygon
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_CurvePolygon) THEN
         RETURN TREAT(SELF AS ST_CurvePolygon);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_CurvePolygon) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_CurvePolygon);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_CurvePolygon().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_CurvePolygon)
   WITH METHOD ST_ToCurvePoly()
   AS ASSIGNMENT

CREATE METHOD ST_ToPolygon()
   RETURNS ST_Polygon
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_Polygon) THEN
         RETURN TREAT(SELF AS ST_Polygon);
      ELSEIF SELF IS OF (ST_CurvePolygon) THEN
         RETURN (SELF AS ST_CurvePolygon).ST_CurvePolyToPoly();
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
```

```
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_CurvePolygon) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_Polygon);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_Polygon().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_Polygon)
   WITH METHOD ST_ToPolygon()
   AS ASSIGNMENT

CREATE METHOD ST_ToTriangle()
   RETURNS ST_Triangle
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_Triangle) THEN
         RETURN TREAT(SELF AS ST_Triangle);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Triangle) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_Triangle);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_Triangle().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_PolyhdrlSurface)
   WITH METHOD ST_ToPolyhdrlSurf()
   AS ASSIGNMENT

CREATE METHOD ST_ToPolyhdrlSurf()
   RETURNS ST_PolyhdrlSurface
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_PolyhdrlSurface) THEN
         RETURN TREAT(SELF AS ST_PolyhdrlSurface);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_PolyhdrlSurface) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_PolyhdrlSurface);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
```

```
      SIGNAL SQLSTATE '2FF16'
         SET MESSAGE_TEXT = 'not an empty set';
   END IF;
   RETURN NEW ST_PolyhdrlSurface().ST_SRID(SELF.ST_SRID());
END

CREATE CAST(ST_Geometry AS ST_PolyhdrlSurface)
   WITH METHOD ST_ToPolyhdrlSurf()
   AS ASSIGNMENT

CREATE METHOD ST_ToTIN()
   RETURNS ST_TIN
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_TIN) THEN
         RETURN TREAT(SELF AS ST_TIN);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_TIN) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_TIN);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_TIN().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_TIN)
   WITH METHOD ST_ToTIN()
   AS ASSIGNMENT

CREATE METHOD ST_ToCompSurface()
   RETURNS ST_CompoundSurface
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_CompoundSurface) THEN
         RETURN TREAT(SELF AS ST_CompoundSurface);
      ELSEIF SELF IS OF (ST_CurvePolygon, ST_Polygon, ST_Triangle, ST_TIN,
      ST_PolyhdrlSurface) THEN
         RETURN NEW ST_CompoundSurfaceARRAY[TREAT(SELF AS ST_Surface)],
            SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_Surface) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_CompoundSurface);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_CompoundSurface ().ST_SRID(SELF.ST_SRID());
   END
```

```
CREATE CAST(ST_Geometry AS ST_CompoundSurface)
   WITH METHOD ST_ToCompSurface()
   AS ASSIGNMENT

CREATE METHOD ST_ToBRepSolid()
   RETURNS ST_BRepSolid
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_BRepSolid) THEN
         RETURN TREAT(SELF AS ST_BRepSolid);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         IF (SELF AS ST_GeomCollection).ST_NumGeometries() = 1 THEN
            IF (SELF AS ST_GeomCollection).ST_GeometryN(1)
                  IS OF (ST_BRepSolid) THEN
               RETURN
                  CAST((SELF AS ST_GeomCollection).ST_GeometryN(1)
                     AS ST_BRepSolid);
            END IF;
         END IF;
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_BRepSolid ().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_BRepSolid)
   WITH METHOD ST_ToBRepSolid()
   AS ASSIGNMENT

CREATE METHOD ST_ToGeomColl()
   RETURNS ST_GeomCollection
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_GeomCollection) THEN
         RETURN TREAT(SELF AS ST_GeomCollection);
      ELSEIF SELF IS OF (ST_Point, ST_Curve, ST_Surface) THEN
         RETURN NEW ST_GeomCollection(ARRAY[SELF], SELF.ST_SRID());
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_GeomCollection().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_GeomCollection)
   WITH METHOD ST_ToGeomColl()
   AS ASSIGNMENT

CREATE METHOD ST_ToMultiPoint()
   RETURNS ST_MultiPoint
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_MultiPoint) THEN
         RETURN TREAT(SELF AS ST_MultiPoint);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         RETURN NEW ST_MultiPoint(
            (SELF AS ST_GeomCollection).ST_Geometries(), SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_Point) THEN
         RETURN NEW ST_MultiPoint(ARRAY[CAST(SELF AS ST_Point)],
            SELF.ST_SRID());
```



```
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_MultiPoint().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_MultiPoint)
   WITH METHOD ST_ToMultiPoint()
   AS ASSIGNMENT

CREATE METHOD ST_ToMultiCurve()
   RETURNS ST_MultiCurve
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_MultiCurve) THEN
         RETURN TREAT(SELF AS ST_MultiCurve);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         RETURN NEW ST_MultiCurve(
            (SELF AS ST_GeomCollection).ST_Geometries(), SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_Curve) THEN
         RETURN NEW ST_MultiCurve(ARRAY[TREAT(SELF AS ST_Curve)],
            SELF.ST_SRID());
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_MultiCurve().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_MultiCurve)
   WITH METHOD ST_ToMultiCurve()
   AS ASSIGNMENT

CREATE METHOD ST_ToMultiLine()
   RETURNS ST_MultiLineString
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_MultiLineString) THEN
         RETURN TREAT(SELF AS ST_MultiLineString);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         RETURN NEW ST_MultiLineString(
            (SELF AS ST_GeomCollection).ST_Geometries(), SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_Curve) THEN
         RETURN NEW
            ST_MultiLineString(ARRAY[CAST(SELF AS ST_LineString)],
               SELF.ST_SRID());
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_MultiLineString().ST_SRID(SELF.ST_SRID());
   END
```



```
CREATE CAST(ST_Geometry AS ST_MultiLineString)
   WITH METHOD ST_ToMultiLine()
   AS ASSIGNMENT

CREATE METHOD ST_ToMultiSurface()
   RETURNS ST_MultiSurface
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_MultiSurface) THEN
         RETURN TREAT(SELF AS ST_MultiSurface);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         RETURN NEW ST_MultiSurface(
            (SELF AS ST_GeomCollection).ST_Geometries(), SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_Surface) THEN
         RETURN NEW ST_MultiSurface(ARRAY[TREAT(SELF AS ST_Surface)],
            SELF.ST_SRID());
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_MultiSurface().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_MultiSurface)
   WITH METHOD ST_ToMultiSurface()
   AS ASSIGNMENT

CREATE METHOD ST_ToMultiPolygon()
   RETURNS ST_MultiPolygon
   FOR ST_Geometry
   BEGIN
      IF SELF IS OF (ST_MultiPolygon) THEN
         RETURN TREAT(SELF AS ST_MultiPolygon);
      ELSEIF SELF IS OF (ST_GeomCollection) THEN
         RETURN NEW ST_MultiPolygon(
            (SELF AS ST_GeomCollection).ST_Geometries(), SELF.ST_SRID());
      ELSEIF SELF IS OF (ST_CurvePolygon) THEN
         RETURN NEW ST_MultiPolygon(ARRAY[CAST(SELF AS ST_Polygon)],
            SELF.ST_SRID());
      END IF;
      IF SELF.ST_IsEmpty() = 0 THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN NEW ST_MultiPolygon().ST_SRID(SELF.ST_SRID());
   END

CREATE CAST(ST_Geometry AS ST_MultiPolygon)
   WITH METHOD ST_ToMultiPolygon()
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_ToPoint()* has no input parameters.

2) For the null-call method *ST_ToPoint()*:

   Case:

   a) If SELF is of type *ST_Point*, then return SELF.

b) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_Point*, then return the element cast to an *ST_Point* value.

c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

d) Otherwise, return an empty set of type *ST_Point* with the spatial reference system identifier set to *SELF.ST_SRID*().

3) Use the method *ST_ToPoint()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_Point* value.

4) The method *ST_ToLineString()* has no input parameters.

5) For the null-call method *ST_ToLineString()*:

Case:

a) If SELF is of type *ST_LineString*, then return SELF.

b) If SELF is of type *ST_CircularString*, *ST_Circle*, *ST_GeodesicString*, *ST_EllipticalCurve*, *ST_NURBSCurve*, *ST_Clothoid*, *ST_SpiralCurve* or *ST_CompoundCurve*, then return *SELF.ST_CurveToLine()*.

c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_Curve*, then return the element cast to an *ST_LineString* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_LineString* with the spatial reference system identifier set to *SELF.ST_SRID()*.

6) Use the method *ST_ToLineString()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_LineString* value.

7) The method *ST_ToCircular()* has no input parameters.

8) For the null-call method *ST_ToCircular()*:

Case:

a) If SELF is of type *ST_CircularString*, then return SELF.

b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_CircularString*, then return the element cast to an *ST_CircularString* value.

c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_CircularString*, then return the element cast to an *ST_CircularString* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_CircularString* with the spatial reference system identifier set to *SELF.ST_SRID()*.

9) Use the method *ST_ToCircular()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_CircularString* value.

10) The method *ST_ToCircle()* has no input parameters.

11) For the null-call method *ST_ToCircle()*:

Case:

a) If SELF is of type *ST_Circle*, then return SELF.

b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_Circle*, then return the element cast to an *ST_*Circle value.

c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_Circle*, then return the element cast to an *ST_Circle* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_Circle* with the spatial reference system identifier set to *SELF.ST_SRID()*.

12) Use the method *ST_ToCircle()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_Circle* value.

13) The method *ST_ToGeodesic()* has no input parameters.

14) For the null-call method *ST_ToGeodesic()*:

Case:

a) If SELF is of type *ST_GeodesicString*, then return SELF.

b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_GeodesicString*, then return the element cast to an *ST_GeodesicString* value.

c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_GeodesicString*, then return the element cast to an *ST_GeodesicString* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_GeodesicString* with the spatial reference system identifier set to *SELF.ST_SRID()*.

15) Use the method *ST_ToGeodesic()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_GeodesicString* value.

16) The method *ST_ToElliptical()* has no input parameters.

17) For the null-call method *ST_ToElliptical()*:

Case:

a) If SELF is of type *ST_EllipticalCurve*, then return SELF.

b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_EllipticalCurve*, then return the element cast to an *ST_EllipticalCurve* value.

c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_EllipticalCurve*, then return the element cast to an *ST_EllipticalCurve* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_EllipticalCurve* with the spatial reference system identifier set to *SELF.ST_SRID()*.

18) Use the method *ST_ToElliptical()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_EllipticalCurve* value.

19) The method *ST_ToNURBSCurve()* has no input parameters.

20) For the null-call method *ST_ToNURBSCurve()*:

Case:

a) If SELF is of type *ST_NURBSCurve*, then return SELF.

b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_NURBSCurve*, then return the element cast to an *ST_NURBSCurve* value.

c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_NURBSCurve*, then return the element cast to an *ST_NURBSCurve* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_NURBSCurve* with the spatial reference system identifier set to *SELF.ST_SRID()*.

21) Use the method *ST_ToNURBSCurve()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_NURBSCurve* value.

22) The method *ST_ToClothoid()* has no input parameters.

23) For the null-call method *ST_ToClothoid()*:

    Case:

    a) If SELF is of type *ST_Clothoid*, then return SELF.

    b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_Clothoid*, then return the element cast to an *ST_Clothoid* value.

    c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_Clothoid*, then return the element cast to an *ST_Clothoid* value.

    d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

    e) Otherwise, return an empty set of type *ST_Clothoid* with the spatial reference system identifier set to *SELF.ST_SRID()*.

24) Use the method *ST_ToClothoid()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_Clothoid* value.

25) The method *ST_ToSpiralCurve()* has no input parameters.

26) For the null-call method *ST_ToSpiralCurve()*:

    Case:

    a) If SELF is of type *ST_SpiralCurve*, then return SELF.

    b) If SELF is of type *ST_CompoundCurve* and SELF has only one element of type *ST_SpiralCurve*, then return the element cast to an *ST_SpiralCurve* value.

    c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_SpiralCurve*, then return the element cast to an *ST_SpiralCurve* value.

    d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

    e) Otherwise, return an empty set of type *ST_SpiralCurve* with the spatial reference system identifier set to *SELF.ST_SRID()*.

27) Use the method *ST_ToSpiralCurve()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_SpiralCurve* value.

28) The method *ST_ToCompound()* has no input parameters.

29) For the null-call method *ST_ToCompound()*:

    Case:

    a) If SELF is of type *ST_CompoundCurve*, then return SELF.

    b) If SELF is of type *ST_LineString*, *ST_CircularString*, *ST_Circle*, *ST_GeodesicString*, *ST_EllipticalCurve*, *ST_NURBSCurve*, *ST_Clothoid* or *ST_SpiralCurve*, then return an *ST_CompoundCurve* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF.

    c) If SELF is of type *ST_GeomCollection* and SELF has only one element of type *ST_Curve*, then return the element cast as an *ST_CompoundCurve* value.

    d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

    e) Otherwise, return an empty set of type *ST_CompoundCurve* with the spatial reference system identifier set to *SELF.ST_SRID()*.

30) Use the method *ST_ToCompound()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_CompoundCurve* value.

31) The method *ST_ToCurvePoly()* has no input parameters.

32) For the null-call method *ST_ToCurvePoly()*:

    Case:

    a) If SELF is of type *ST_CurvePolygon*, then return SELF.

    b) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_CurvePolygon*, then return the element cast as an *ST_CurvePolygon* value.

    c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

    d) Otherwise, return an empty set of type *ST_CurvePolygon* with the spatial reference system identifier set to *SELF.ST_SRID()*.

33) Use the method *ST_ToCurvePoly()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_CurvePolygon* value.

34) The method *ST_ToPolygon()* has no input parameters.

35) For the null-call method *ST_ToPolygon()*:

    Case:

    a) If SELF is of type *ST_Polygon*, then return SELF.

    b) If SELF is of type *ST_CurvePolygon*, then return *SELF.ST_CurvePolyToPoly()*.

    c) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_CurvePolygon*, then return the element cast as an *ST_Polygon* value.

    d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

    e) Otherwise, return an empty set of type *ST_Polygon* with the spatial reference system identifier set to *SELF.ST_SRID()*.

36) Use the method *ST_ToPolygon()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_Polygon* value.

37) The method *ST_ToTriangle()* has no input parameters.

38) For the null-call method *ST_ToTriangle()*:

    Case:

    a) If SELF is of type *ST_Triangle*, then return SELF.

    b) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_Triangle*, then return the element cast as an *ST_Triangle* value.

    c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

    d) Otherwise, return an empty set of type *ST_Triangle* with the spatial reference system identifier set to *SELF.ST_SRID()*.

39) Use the method *ST_ToTriangle()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_Triangle* value.

40) The method *ST_ToPolyhdrlSurf()* has no input parameters.

41) For the null-call method *ST_ToPolyhdrlSurf()*:

    Case:

    a) If SELF is of type *ST_PolyhdrlSurface*, then return SELF.

    b) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_PolyhdrlSurface*, then return the element cast as an *ST_PolyhdrlSurface* value.

c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

d) Otherwise, return an empty set of type *ST_PolyhdrlSurface* with the spatial reference system identifier set to *SELF.ST_SRID()*.

42) Use the method *ST_ToPolyhdrlSurf()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_PolyhdrlSurface* value.

43) The method *ST_ToTIN()* has no input parameters.

44) For the null-call method *ST_ToTIN()*:

Case:

a) If SELF is of type *ST_TIN*, then return SELF.

b) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_TIN*, then return the element cast as an *ST_TIN* value.

c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

d) Otherwise, return an empty set of type *ST_TIN* with the spatial reference system identifier set to *SELF.ST_SRID()*.

45) Use the method *ST_ToTIN()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_TIN* value.

46) The method *ST_ToCompSurface()* has no input parameters.

47) For the null-call method *ST_ToCompSurface()*:

Case:

a) If SELF is of type *ST_CompoundSurface*, then return SELF.

b) If SELF is of type *ST_CurvePolygon*, *ST_Polygon*, *ST_Triangle*, *ST_TIN*, or *ST_PolyhdrlSurface* then return an *ST_CompoundCurve* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF.

c) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_Surface*, then return the element cast as an *ST_CompoundSurface* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_CompoundSurface* with the spatial reference system identifier set to *SELF.ST_SRID()*.

48) Use the method *ST_ToCompSurfacemethod*"*()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_CompoundSurface* value.

49) The method *ST_ToBRepSolid()* has no input parameters.

50) For the null-call method *ST_ToBRepSolid()*:

Case:

a) If SELF is of type *ST_BRepSolid*, then return SELF.

b) If SELF is of type *ST_GeomCollection* and SELF has one element of type *ST_BRepSolid*, then return the element cast as an *ST_BRepSolid* value.

c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

d) Otherwise, return an empty set of type *ST_BRepSolid* with the spatial reference system identifier set to *SELF.ST_SRID()*.

51) Use the method *ST_ToBRepSolid*method"*()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_BRepSolid* value.

52) The method *ST_ToGeomColl()* has no input parameters.

53) For the null-call method *ST_ToGeomColl()*:

Case:

a) If SELF is of type *ST_GeomCollection*, then return SELF.

b) If SELF is of type *ST_Point*, *ST_Curve* or *ST_Surface*, then return an *ST_GeomCollection* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF.

c) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

d) Otherwise, return an empty set of type *ST_GeomCollection* with the spatial reference system identifier set to *SELF.ST_SRID()*.

54) Use the method *ST_ToGeomColl()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_GeomCollection* value.

55) The method *ST_ToMultiPoint()* has no input parameters.

56) For the null-call method *ST_ToMultiPoint()*:

Case:

a) If SELF is of type *ST_MultiPoint*, then return SELF.

b) If SELF is of type *ST_GeomCollection*, then return an *ST_MultiPoint* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing the elements of SELF.

c) If SELF is of type *ST_Point*, then return an *ST_MultiPoint* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF, cast to an *ST_Point* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_MultiPoint* with the spatial reference system identifier set to *SELF.ST_SRID()*.

57) Use the method *ST_ToMultiPoint()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_MultiPoint* value.

58) The method *ST_ToMultiCurve()* has no input parameters.

59) For the null-call method *ST_ToMultiCurve()*:

Case:

a) If SELF is of type *ST_MultiCurve*, then return SELF.

b) If SELF is of type *ST_GeomCollection*, then return an *ST_MultiCurve* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing the elements of SELF.

c) If SELF is of type *ST_Curve* then return an *ST_MultiCurve* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF, treated aw an *ST_Curve* value.

d) If SELF is not an empty set value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_MultiCurve* with the spatial reference system identifier set to *SELF.ST_SRID()*.

60) Use the method *ST_ToMultiCurve()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_MultiCurve* value.

61) The method *ST_ToMultiLine()* has no input parameters.

62) For the null-call method *ST_ToMultiLine()*:

Case:

a) If SELF is of type *ST_MultiLineString*, then return SELF.

b) If SELF is of type *ST_GeomCollection*, then return an *ST_MultiLineString* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing the elements of SELF.

c) If SELF is of type *ST_LineString*, then return an *ST_MultiLineString* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF, cast to an *ST_LineString* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_MultiLineString* with the spatial reference system identifier set to *SELF.ST_SRID()*.

63) Use the method *ST_ToMultiLine()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_MultiLineString* value.

64) The method *ST_ToMultiSurface()* has no input parameters.

65) For the null-call method *ST_ToMultiSurface()*:

Case:

a) If SELF is of type *ST_MultiSurface*, then return SELF.

b) If SELF is of type *ST_GeomCollection*, then return an *ST_MultiSurface* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing the elements of SELF.

c) If SELF is of type *ST_Surface*, then return an *ST_MultiSurface* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF, treated as an *ST_Surface* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_MultiSurface* with the spatial reference system identifier set to *SELF.ST_SRID()*.

66) Use the method *ST_ToMultiSurface()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_MultiSurface* value.

67) The method *ST_ToMultiPolygon()* has no input parameters.

68) For the null-call method *ST_ToMultiPolygon()*:

Case:

a) If SELF is of type *ST_MultiPolygon*, then return SELF.

b) If SELF is of type *ST_GeomCollection*, then return an *ST_MultiPolygon* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing the elements of SELF.

c) If SELF is of type *ST_Polygon*, then return an *ST_MultiPolygon* value with the spatial reference system identifier set to *SELF.ST_SRID()* and containing one element, SELF cast to an *ST_Polygon* value.

d) If SELF is not an empty set, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

e) Otherwise, return an empty set of type *ST_MultiPolygon* with the spatial reference system identifier set to *SELF.ST_SRID()*.

69) Use the method *ST_ToMultiPolygon()* to define an implicitly invocable cast function to cast an *ST_Geometry* value to an *ST_MultiPolygon* value.

5.1.56 ST_WKTToSQL Method

Purpose

Return an ST_Geometry value for a given well-known text representation.

Definition

```
CREATE METHOD ST_WKTToSQL
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The method *ST_WKTToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awkt*.

2) The parameter *awkt* is the well-known text representation of an *ST_Geometry* value. If *awkt* is not producible in the BNF for <well-known text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

3) The null-call method *ST_WKTToSQL(CHARACTER LARGE OBJECT)* returns an *ST_Geometry* value represented by *awkt.*

5.1.57 ST_AsText Method

Purpose

Return the well-known text representation of an ST_Geometry value.

Definition

```
CREATE METHOD ST_AsText()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxGeometryAsText)
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The method *ST_AsText()* has no input parameters.

2) The null-call method *ST_AsText()* returns a CHARACTER LARGE OBJECT value containing the well-known text representation of SELF. Values shall be produced in the BNF for <well-known text representation>.

5.1.58 ST_WKBToSQL Method

Purpose

Return an ST_Geometry value for a given well-known binary representation.

Definition

```
CREATE METHOD ST_WKBToSQL
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The method *ST_WKBToSQL(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The parameter *awkb* is the well-known binary representation of an *ST_Geometry* value. If *awkb* is not producible in the BNF for <well-known binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
3) The null-call method *ST_WKBToSQL(BINARY LARGE OBJECT)* returns an *ST_Geometry* value represented by *awkb*.

5.1.59 ST_AsBinary Method

Purpose

Return the well-known binary representation of an ST_Geometry value.

Definition

```
CREATE METHOD ST_AsBinary()
   RETURNS BINARY LARGE OBJECT(ST_MaxGeometryAsBinary)
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The method *ST_AsBinary()* has no input parameters.
2) The null-call method *ST_AsBinary()* returns a BINARY LARGE OBJECT value containing the well-known binary representation of SELF. Values shall be produced in the BNF for <well-known binary representation>.

5.1.60 ST_GMLToSQL Method

Purpose

Return an ST_Geometry value for a given GML representation.

Definition

```
CREATE METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Geometry
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The parameter *agml* is the GML representation of an *ST_Geometry* value. If *agml* does not contain an XML element as defined in Table 14 — Mapping between ST_Geometry values and GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

Table 14 — Mapping between ST_Geometry values and GML representation

XML Element in the GML representation	ST_Geometry values
Point	ST_Point
LineString	ST_LineString
LineStringSegment	ST_LineString
Circle	ST_Circle
CircleByCenterPoint	ST_Circle
Arc	ST_CircularString wth 3 points
ArcString	ST_CircularString
ArcByBulge	ST_CircularString wth 3 points
ArcStringByBulge	ST_CircularString
ArcByCenterPoint	ST_CircularString wth 3 points
Geodesic	ST_GeodesicString wth 2 points
GeodesicString	ST_GeodesicString
EllipticalCurve	ST_EllipticalCurve
Ellipse	ST_EllipticalCurve
BSpline	ST_NURBSCurve
Clothoid	ST_Clothoid
SpiralCurve	ST_SpiralCurve
CompositeCurve	ST_CompoundCurve
PolygonPatch	ST_CurvePolygon
Polygon	ST_Polygon
Triangle	ST_Triangle
PolyhedralSurface	ST_PolyhdrlSurface
Tin from GML 3.2.1	ST_TIN
Tin from GML 3.3	ST_TIN
CompositeSurface	ST_CompoundSurface

XML Element in the GML representation	ST_Geometry values
Solid	ST_BRepSolid
MultiGeometry	ST_GeomCollection
MultiPoint	ST_MultiPoint
MultiCurve	ST_MultiCurve
MultiLineString	ST_MultiLineString
MultiSurface	ST_MultiSurface
MultiPolygon	ST_MultiPolygon

3) The x coordinate value of an *ST_Point* value is represented as either the X XML element of a coord XML element, the first coordinate value of a coordinates XML element or the first coordinate value of a pos XML element.

4) The y coordinate value of an *ST_Point* value is represented as either the Y XML element of a coord XML element, the second coordinate value of a coordinates XML element or the second coordinate value of a pos XML element.

5) The z coordinate value of an *ST_Point* value is represented as either the Z XML element of a coord XML element, the third coordinate value of coordinates XML element or the third coordinate value of a pos XML element.

   Case:

   a) If all the coord XML elements contain the Z XML element, all the coordinates XML elements contain at least 3 coordinate values and the all the pos XML elements have at least 3 coordinate values, then all the *ST_Point* values contained in the resulting *ST_Geometry* value will have x, y, and z coordinate values.

   b) Otherwise, all the *ST_Point* values contained in the resulting *ST_Geometry* value will have only x and y coordinate values.

6) The *ST_Point* values contained in the resulting *ST_Geometry* value will not have m coordinate values.

7) Let *S* be the spatial reference system identifier for the resulting *ST_Geometry* value.

   Case:

   a) If the srsname XML attribute is not specified, then set *S* to 0 (zero).

   b) Otherwise,

      Case:

      i) If the value of srsname XML attribute is producible in the BNF for <spatial reference system>, then

         1) Set *SRT* to the spatial reference system text in the srsname XML attribute.

         2) Select the row in the SPATIAL_REF_SYS view where the SRTEXT column is equal to *SRT*.

            Case:

            A) If the row is not found, then the following exception condition is raised: *SQL/MM Spatial Exception – unknown spatial reference system*.

            B) Otherwise, set *S* to the value of the SRID column in the returned row.

      ii) If the value of the srsname XML attribute is in the form: *ON*:*OI* where *ON* is the organization name and *OI* is organization assigned identifier, then

         1) Let *AN* be the organization name *ON* and *AI* be the organization assigned identifier *OI*.

         2) Select the row in the SPATIAL_REF_SYS view where the AUTH_NAME column is equal to *AN* and AUTH_ID column is equal to *AI*.

            Case:

A) If the row is not found, then the following exception condition is raised: *SQL/MM Spatial Exception – unknown spatial reference system*.

B) Otherwise, set *S* to the value of the SRID column in the returned row.

iii) Otherwise, the following exception condition is raised: *SQL/MM Spatial Exception – unknown spatial reference system*.

8) The null-call method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* returns an *ST_Geometry* value represented by *agml*.

9) *ST_GMLToSQL* returns an *ST_Polygon* value for a GML Polygon XML element. To instead obtain an *ST_CurvePolygon* value, use the *ST_CurvePolygon(CHARACTER LARGE OBJECT)* or *ST_CurvePolygon(CHARACTER LARGE OBJECT, INTEGER)* constructor.

10) *ST_GMLToSQL* returns an *ST_CurvePolygon* value for a GML PolygonPatch XML element. To instead obtain an *ST_Polygon* value, use the *ST_Polygon(CHARACTER LARGE OBJECT)* or *ST_Polygon(CHARACTER LARGE OBJECT, INTEGER)* constructor. To obtain an *ST_PolyhdrlSurface* value, use the *ST_PolyhdrlSurface (CHARACTER LARGE OBJECT)* or *ST_PolyhdrlSurface (CHARACTER LARGE OBJECT, INTEGER)* constructor.

5.1.61 ST_AsGML Method

Purpose

Return the GML representation of an ST_Geometry value.

Definition

```
CREATE METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML)
   FOR ST_Geometry
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The method *ST_AsGML()* has no input parameters.
2) The null-call method *ST_AsGML()* returns a CHARACTER LARGE OBJECT value containing a GML representation. The instantiable subtypes of ST_Geometry are mapped to XML elements in the GML representation as defined in Table 14 — Mapping between ST_Geometry values and GML representation.
3) The srsname XML attribute of the XML element identifies its spatial reference system. Select the row in the SPATIAL_REF_SYS view where the *srid* is equal to *SELF.ST_SRID()*. For the selected row, let *AN* be the value of the AUTH_NAME column, *AI* be the value of the AUTH_ID column and *SRT* be the value of the SRTEXT column.

   Case:

   a) If the *AN* is not the null value and *AI* is not the null value then the srsname XML attribute attribute is specified as:

      srsname='*AN*:*AI*'

   b) Otherwise, the srsname XML attribute is specified as:

      srsname='*SRT*'
4) If SELF.*ST_IsMeasured()* is equal to 1 (one), then the GML representation does not contain the m coordinate values.

5.1.62 ST_GeomFromText Functions

Purpose

Return an ST_Geometry value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Geometry value.

Definition

```
CREATE FUNCTION ST_GeomFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeomFromText(awkt, 0)

CREATE FUNCTION ST_GeomFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_GeomFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_GeomFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_GeomFromText(awkt, 0)*.
3) The function *ST_GeomFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeomFromText(CHARACTER LARGE OBJECT, INTEGER)*:
   a) The parameter *awkt* is the well-known text representation of an *ST_Geometry* value. If *awkt* is not producible in the BNF for <well-known text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Return an *ST_Geometry* value represented by *awkt* with the spatial reference system identifier set to *ansrid*.

5.1.63 ST_GeomFromWKB Functions

Purpose

Return an ST_Geometry value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Geometry value.

Definition

```
CREATE FUNCTION ST_GeomFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeomFromWKB(awkb, 0)

CREATE FUNCTION ST_GeomFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_GeomFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_GeomFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_GeomFromWKB(awkb, 0)*.
3) The function *ST_GeomFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeomFromWKB(BINARY LARGE OBJECT, INTEGER)*:
   a) The parameter *awkb* is the well-known binary representation of an *ST_Geometry* value. If *awkb* is not producible in the BNF for <well-known binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Return an *ST_Geometry* value represented by *awkb* with the spatial reference system identifier set to *ansrid*.

5.1.64 ST_GeomFromGML Functions

Purpose

Return an ST_Geometry value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Geometry.

Definition

```
CREATE FUNCTION ST_GeomFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeomFromGML(agml, 0)

CREATE FUNCTION ST_GeomFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_GeomFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_GeomFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_GeomFromGML(agml, 0)*.
3) The function *ST_GeomFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeomFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain an XML element as defined in Table 14 — Mapping between ST_Geometry values and GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Return an *ST_Geometry* value represented by *agml* with the spatial reference system identifier set to *ansrid*.
   c) The x coordinate value of an *ST_Point* value is represented as either the X XML element of a coord XML element, the first coordinate value of a coordinates XML element or the first coordinate value of a pos XML element.

d) The y coordinate value of an *ST_Point* value is represented as either the Y XML element of a coord XML element, the second coordinate value of a coordinates XML element or the second coordinate value of a pos XML element.

e) The z coordinate value of an *ST_Point* value is represented as either the Z XML element of a coord XML element, the third coordinate value of a coordinates XML element or the third coordinate value of a pos XML element.

   Case:

   i) If all the coord XML elements contain the Z XML element, all the coordinates XML elements contain at least 3 coordinate values and all the pos XML elements contain at least 3 pos values, then all the *ST_Point* values contained in the resulting *ST_Geometry* value will have x, y, and z coordinate values.

   ii) Otherwise, all the *ST_Point* values contained in the resulting *ST_Geometry* value will have only x and y coordinate values.

f) The *ST_Point* values contained in the resulting *ST_Geometry* value will not have m coordinate values.

5.1.65 ST_Geometry Ordering Definition

Purpose

Provide the equals only ordering definition for the ST_Geometry type.

Definition

```
CREATE FUNCTION ST_OrderingEquals
   (ageometry ST_Geometry,
    anothergeometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      IF ageometry.ST_Is3D() = 1 AND anothergeometry.ST_Is3D() = 1 THEN
         IF ageometry.ST_3DEquals(anothergeometry) = 1 THEN
            RETURN 0;
         ELSE
            RETURN 1;
         END IF;
      ELSE
         IF ageometry.ST_Equals(anothergeometry) = 1 THEN
            RETURN 0;
         ELSE
            RETURN 1;
         END IF;
      END IF;
   END

CREATE ORDERING FOR ST_Geometry
   EQUALS ONLY BY RELATIVE WITH
      FUNCTION ST_OrderingEquals(ST_Geometry, ST_Geometry)
```

Description

1) The function *ST_OrderingEquals(ST_Geometry, ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*,
   b) an *ST_Geometry* value *anothergeometry*.
2) For the null-call function *ST_OrderingEquals(ST_Geometry, ST_Geometry)*:

   Case:
   a) If *ageometry.ST_Is3D()* is 1 (one) and *anothergeometry.ST_Is3D()* is 1 (one), then:

      Case:
      i) If the value expression: *ageometry.ST_3DEquals(anothergeometry)* is 1 (one), then return 0 (zero).
      ii) Otherwise, return 1 (one).
   b) Otherwise,

      Case:
      i) If the value expression: *ageometry.ST_Equals(anothergeometry)* is 1 (one), then return 0 (zero).
      ii) Otherwise, return 1 (one).
3) Use the function *ST_OrderingEquals(ST_Geometry, ST_Geometry)* to define ordering for the *ST_Geometry* type.

5.1.66 SQL Transform Functions

Purpose

Define SQL transform functions for the ST_Geometry type.

Definition

```
CREATE TRANSFORM FOR ST_Geometry
   ST_WellKnownText
      (TO SQL WITH METHOD ST_WKTToSQL
         (CHARACTER LARGE OBJECT(ST_MaxGeometryAsText)),
       FROM SQL WITH METHOD ST_AsText())
   ST_WellKnownBinary
      (TO SQL WITH METHOD ST_WKBToSQL
         (BINARY LARGE OBJECT(ST_MaxGeometryAsBinary)),
       FROM SQL WITH METHOD ST_AsBinary())
   ST_GML
      (TO SQL WITH METHOD ST_GMLToSQL
         (CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML)),
       FROM SQL WITH METHOD ST_AsGML())
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) Use the method *ST_WKTToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsText()* to define the transform group *ST_WellKnownText*.
2) Use the method *ST_WKBToSQL(BINARY LARGE OBJECT)* and the method ST_AsBinary*()* to define the transform group *ST_WellKnownBinary*.
3) Use the method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsGML()* to define the transform group *ST_GML*.

5.1.67 <well-known text representation>

Purpose

This subclause contains the definition of <well-known text representation>.

Description

1) The well-known text representation of an *ST_Geometry* value is defined by the following BNF for <well-known text representation>.

```
<well-known text representation> ::=
     <point text representation>
   | <curve text representation>
   | <surface text representation>
   | <solid text representation>
   | <collection text representation>

<point text representation> ::=
     POINT [ <z m> ] <point text>

<curve text representation> ::=
     <linestring text representation>
   | <circularstring text representation>
   | <circle text representation>
   | <geodesic text representation>
   | <elliptical text representation>
   | <nurbs text representation>
   | <clothoid text representation>
   | <spiral text representation>
   | <compoundcurve text representation>

<linestring text representation> ::=
     LINESTRING [ <z m> ] <linestring text body>

<circularstring text representation> ::=
     CIRCULARSTRING [ <z m> ] <circularstring text>

<circle text representation> ::=
     CIRCLE [ <z m> ] <circle text>

<geodesic text representation> ::=
     GEODESICSTRING [ <z m> ] <geodesic text>

<elliptical text representation> ::=
     ELLIPTICALCURVE [ <z m> ] <elliptical text>

<nurbs text representation> ::=
     NURBSCURVE [ <z m> ] <nurbs text>

<clothoid text representation> ::=
     CLOTHOID [ <z m> ] <clothoid text>

<spiral text representation> ::=
     SPIRALCURVE [ <z m> ] <spiral text>

<compoundcurve text representation> ::=
     COMPOUNDCURVE [ <z m> ] <compoundcurve text>

<surface text representation> ::=
     <curvepolygon text representation>
   | <polyhedralsurface text representation>
   | <compoundsurface text representation>

<curvepolygon text representation> ::=
     CURVEPOLYGON [ <z m> ] <curvepolygon text body>
   | <polygon text representation>

<polygon text representation> ::=
```

```
     POLYGON [ <z m> ] <polygon text body>
   | <triangle text representation>

<triangle text representation> ::=
     TRIANGLE [ <z m> ] <triangle text body>

<polyhedralsurface text representation> ::=
     POLYHEDRALSURFACE [ <z m> ] <polyhedralsurface text body>
   | <tin text representation>

<tin text representation> ::=
     TIN [ <z m> ] <tin text body>

<compoundsurface text representation> ::=
     COMPOUNDSURFACE [ <z m> ] <compoundsurface text>

<solid text representation> ::=
     <brepsolid text representation>

<brepsolid text representation> ::=
     BREPSOLID Z <brepsolid text>

<collection text representation> ::=
     <multipoint text representation>
   | <multicurve text representation>
   | <multisurface text representation>
   | <geometrycollection text representation>

<multipoint text representation> ::=
     MULTIPOINT [ <z m> ] <multipoint text>

<multicurve text representation> ::=
     MULTICURVE [ <z m> ] <multicurve text>
   | <multilinestring text representation>

<multilinestring text representation> ::=
     MULTILINESTRING [ <z m> ] <multilinestring text>

<multisurface text representation> ::=
     MULTISURFACE [ <z m> ] <multisurface text>
   | <multipolygon text representation>

<multipolygon text representation> ::=
     MULTIPOLYGON [ <z m> ] <multipolygon text>

<geometrycollection text representation> ::=
     GEOMETRYCOLLECTION [ <z m> ] <geometrycollection text>

<linestring text body> ::=
     <linestring text>

<curvepolygon text body> ::=
     <curvepolygon text>

<polygon text body> ::=
     <polygon text>

<triangle text body> ::=
     <triangle text>

<polyhedralsurface text body> ::=
     <polyhedralsurface text>

<tin text body> ::=
     <tin text>

<point text> ::=
     <empty set>
   | <left paren> <point> <right paren>

<point> ::= <x> <y> [ <z> ] [ <m> ]
```

```
<x> ::= <number>

<y> ::= <number>

<z> ::= <number>

<m> ::= <number>

<linestring text> ::=
     <empty set>
   | <left paren> <point>
        { <comma> <point> }... <right paren>

<circularstring text> ::=
     <empty set>
   | <left paren> <point>
        { <comma> <point> }... <right paren>

<circle text> ::=
     <empty set>
   | <left paren> <point>
        { <comma> <point> }... <right paren>

<geodesic text> ::=
     <empty set>
   | <left paren> <point>
        { <comma> <point> }... <right paren>

<elliptical text> ::=
     <empty set>
   | <left paren> <referencelocation text representation>
        <comma> <uaxislength text representation>
        <comma> <vaxislength text representation>
        <comma> <startangle text representation>
        <comma> <endangle text representation>
      [ <comma> <startm text representation>
        <comma> <endm text representation> ] <right paren>

<referencelocation text representation> ::=
     REFERENCELOCATION <affineplacement text representation>

<uaxislength text representation> ::=
     UAXISLENGTH <length text>

<vaxislength text representation> ::=
     VAXISLENGTH <length text>

<startangle text representation> ::=
     STARTANGLE <angle text>

<endangle text representation> ::=
     ENDANGLE <angle text>

<angle text> ::=
     !! See Subclause 15.1.21 “<angle text representation>”

<startm text representation> ::=
     STARTM <number>

<endm text representation> ::=
     ENDM <number>

<length text> ::=
     <empty set>
   | <number>

<affineplacement text representation> ::=
     AFFINEPLACEMENT [ <just z> ] <affineplacement text>

<affineplacement text> ::=
```

```
     <empty set>
   | <left paren> <location text representation>
        <comma> <referencedirections text representation> <right paren>

<location text representation> ::=
     LOCATION [ <just z> ] <point text>

<referencedirections text representation> ::=
     REFERENCEDIRECTIONS <referencedirections text>

<referencedirections text> ::=
     <empty set>
   | <left paren> <vector text representation>
        { <comma> <vector text representation> }... <right paren>

<vector text representation> ::=
     !! See Subclause 16.2.22, "<well-known text representation>"

<nurbs text> ::=
     <empty set>
   | <left paren> <degree text representation>
        <comma> <controlpoints text representation>
        <comma> <knots text representation>
      [ <comma> <startm text representation>
        <comma> <endm text representation> ] <right paren>

<degree text representation> ::=
     DEGREE <signed integer>

<controlpoints text representation> ::=
     CONTROLPOINTS [ <just z> ] <controlpoints text>

<knots text representation> ::=
     KNOTS <knots text>

<controlpoints text> ::=
     <empty set>
   | <left paren> <nurbspoint text representation>
       { <comma> <nurbspoint text representation> }... <right paren>

<knots text> ::=
     <empty set>
   | <left paren> <knot text representation>
        { <comma> <knot text representation> }... <right paren>

<nurbspoint text representation> ::=
     NURBSPOINT <nurbspoint text>

<nurbspoint text> ::=
     <empty set>
   | <left paren> <weightedpoint text representation>
        <comma> <weight text representation> <right paren>

<weightedpoint text representation> ::=
     WEIGHTEDPOINT [ <just z> ] <point text>

<weight text representation> ::=
     WEIGHT <number>

<knot text representation> ::=
     KNOT <knot text>

<knot text> ::=
     <empty set>
   | <left paren> <value text representation>
        <comma> <multiplicity text representation> <right paren>

<value text representation> ::=
     VALUE <number>
```

```
<multiplicity text representation> ::=
     MULTIPLICITY <signed integer>

<clothoid text> ::=
     <empty set>
   | <left paren> <referencelocation text representation>
        <comma> <scalefactor text representation>
        <comma> <startdistance text representation>
        <comma> <enddistance text representation>
      [ <comma> <startm text representation>
        <comma> <endm text representation> ] <right paren>

<scalefactor text representation> ::=
     SCALEFACTOR <scalefactor text>

<startdistance text representation> ::=
     STARTDISTANCE <distance text>

<enddistance text representation> ::=
     ENDDISTANCE <distance text>

<scalefactor text> ::=
     <empty set>
   | <number>

<distance text> ::=
     <empty set>
   | <number>

<spiral text> ::=
     <empty set>
   | <left paren> <referencelocation text representation>
        <comma> <spirallength text representation>
        <comma> <startcurvature text representation>
        <comma> <endcurvature text representation>
        <comma> <spiraltype text representation>
      [ <comma> <startm text representation>
        <comma> <endm text representation> ] <right paren>

<spirallength text representation> ::=
     LENGTH <spirallength text>

<startcurvature text representation> ::=
     STARTCURVATURE <curvature text>

<endcurvature text representation> ::=
     ENDCURVATURE <curvature text>

<spiraltype text representation> ::=
     SPIRALTYPE <spiraltype text>

<spirallength text> ::=
     <empty set>
   | <number>

<curvature text> ::=
     <empty set>
   | <number>

<spiraltype text> ::=
     <empty set>
   | <letters>

<compoundcurve text> ::=
     <empty set>
   | <left paren> <curve text>
        { <comma> <curve text> }... <right paren>

<curve text> ::=
```

```
      <linestring text body>
    | <circularstring text representation>
    | <circle text representation>
    | <geodesic text representation>
    | <elliptical text representation>
    | <nurbs text representation>
    | <clothoid text representation>
    | <spiral text representation>
    | <compoundcurve text representation>

<ring text> ::=
      <linestring text body>
    | <circularstring text representation>
    | <circle text representation>
    | <geodesic text representation>
    | <elliptical text representation>
    | <nurbs text representation>
    | <clothoid text representation>
    | <spiral text representation>
    | <compoundcurve text representation>

<surface text> ::=
      CURVEPOLYGON <curvepolygon text body>
    | <polygon text body>
    | TRIANGLE <triangle text body>
    | POLYHEDRALSURFACE <polyhedralsurface text body>
    | TIN <tin text body>
    | COMPOUNDSURFACE <compoundsurface text body>

<curvepolygon text> ::=
      <empty set>
    | <left paren> <ring text>
         { <comma> <ring text> }... <right paren>

<polygon text> ::=
      <empty set>
    | <left paren> <linestring text>
         { <comma> <linestring text> }... <right paren>

<triangle text> ::=
      <empty set>
    | <left paren> <point> <comma> <point> <comma> <point> <right paren>

<polyhedralsurface text> ::=
      <empty set>
    | <left paren> PATCHES <polygonpatches text> <right paren>

<polygonpatches text> ::=
      <left paren> <polygon text representation>
         { <comma> <polygon text representation> }... <right paren>

<tin text> ::=
      <empty set>
    | <left paren> PATCHES <trianglepatches text>
         [ ELEMENTS <tinelement list> ]
         [ <maxsidelength> ]
         <right paren>

<trianglepatches text> ::=
      <left paren> <triangle text body>
         { <comma> <triangle text body> }... <right paren>

<tinelement list> ::=
      <left paren> <tinelementtype text>
         { <comma> <tinelementtype text> }... <right paren>
```

```
<tinelementtype text> ::=
     <randompoints representation>
   | <groupspot representation>
   | <boundary representation>
   | <breakline representation>
   | <softbreak representation>
   | <controlcontour representation>
   | <breakvoid representation>
   | <drapevoid representation>
   | <void representation>
   | <hole representation>
   | <stopline representation>

<randompoints representation> ::=
     POINTS <elementlabel text> <multipoint text representation>

<groupspot representation> ::=
     GROUPSPOT <elementlabel text> <multipoint text representation>

<boundary representation> ::=
     BOUNDARY <elementlabel text> <polygon text representation>

<breakline representation> ::=
     BREAKLINE <elementlabel text> <linestring text representation>

<softbreak representation> ::=
     SOFTBREAK <elementlabel text> <linestring text representation>

<controlcontour representation> ::=
     CONTROLCONTOUR <elementlabel text> <linestring text representation>

<breakvoid representation> ::=
     BREAKVOID <elementlabel text> <polygon text representation>

<drapevoid representation> ::=
     DRAPEVOID <elementlabel text> <polygon text representation>

<void representation> ::=
     VOID <elementlabel text> <polygon text representation>

<hole representation> ::=
     HOLE <elementlabel text> <polygon text representation>

<stopline representation ::=
     STOPLINE <elementlabel text> <linestring text representation>

<elementlabel text> ::=
     [ ID <element id> ] [ TAG <element tag> ]

<compoundsurface text> ::=
     <empty set>
   | <left paren> <surface text>
        { <comma> <surface text> }... <right paren>

<brepsolid text> ::=
     <empty set>
   | <left paren> <shell text>
        { <comma> <shell text> }... <right paren>

<shell text> ::=
     POLYHEDRALSURFACE <polyhedralsurface text body>
   | COMPOUNDSURFACE <compoundsurface text body>

<multipoint text> ::=
     <empty set>
   | <left paren> <point text>
        { <comma> <point text > }... <right paren>

<multicurve text> ::=
```

```
      <empty set>
    | <left paren> <curve text>
         { <comma> <curve text> }... <right paren>

<multilinestring text> ::=
      <empty set>
    | <left paren> <linestring text body>
         { <comma> <linestring text body> }... <right paren>

<multisurface text> ::=
      <empty set>
    | <left paren> <surface text>
         { <comma> <surface text> }... <right paren>

<multipolygon text> ::=
      <empty set>
    | <left paren> <polygon text body>
         { <comma> <polygon text body> }... <right paren>

<geometrycollection text> ::=
      <empty set>
    | <left paren> <well-known text representation>
         { <comma> <well-known text representation> }... <right paren>

<empty set> ::= EMPTY

<z m> ::=
      ZM
    | Z
    | M

<just z> ::=
      Z

<element id> ::=
      <signed integer>

<element tag> ::=
      <double quote> <letters> <double quote>

<maxsidelength> ::=
      MAXSIDELENGTH <number>

<double quote> ::=
      !! See Subclause 5.1, "<SQL terminal character>", in
         Part 2 of ISO/IEC 9075

<letters> ::= <letter>...

<letter> ::=
      <simple Latin letter>
    | <digit>
    | <special>


<simple Latin letter> ::=
      !! See Subclause 5.1, "<SQL terminal character>", in
         Part 2 of ISO/IEC 9075

<digit> ::=
      !! See Subclause 5.1, "<SQL terminal character>", in
         Part 2 of ISO/IEC 9075

<special> ::=
      <left paren>
    | <right paren>
    | <minus sign>
    | <underscore>
    | <period>
```

```
    | <quote>
    | <space>

<left paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<minus sign> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<underscore> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<period> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<quote> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<space> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<comma> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<number> ::=
      <exact numeric literal>
    | <approximate numeric literal>

<exact numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<approximate numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<signed integer> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075
```

a) Case:

   i) If <well-known text representation> immediately contains a <point text representation>, then <well-known text representation> produces an *ST_Point* value specified by the immediately contained <point text representation>.

   ii) If <well-known text representation> immediately contains a <curve text representation>, then <well-known text representation> produces an *ST_Curve* value specified by the immediately contained <curve text representation>.

   iii) If <well-known text representation> immediately contains a <surface text representation>, then <well-known text representation> produces an *ST_Surface* value specified by the immediately contained <surface text representation>.

   iv) If <well-known text representation> immediately contains a <solid text representation>, then <well-known text representation> produces an *ST_Solid* value specified by the immediately contained <solid text representation>.

   v) Otherwise, <well-known text representation> produces an *ST_GeomCollection* value specified by the immediately contained <collection text representation>.

b) <point text representation> is the well-known text representation for an *ST_Point* value that is produced by <point text>.

c) Case:

   i) If <curve text representation> immediately contains a <linestring text representation>, then <curve text representation> produces an *ST_LineString* value specified by the immediately contained <linestring text representation>.

   ii) If <curve text representation> immediately contains a <circularstring text representation>, then <curve text representation> produces an *ST_CircularString* value specified by the immediately contained <circularstring text representation>.

   iii) If <curve text representation> immediately contains a <circle text representation>, then <curve text representation> produces an *ST_Circle* value specified by the immediately contained <circle text representation>.

   iv) If <curve text representation> immediately contains a <geodesic text representation>, then <curve text representation> produces an *ST_GeodesicString* value specified by the immediately contained <geodesic text representation>.

   v) If <curve text representation> immediately contains an <elliptical text representation>, then <curve text representation> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical text representation>.

   vi) If <curve text representation> immediately contains a <nurbs text representation>, then <curve text representation> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs text representation>.

   vii) If <curve text representation> immediately contains a <clothoid text representation>, then <curve text representation> produces an *ST_Clothoid* value specified by the immediately contained <clothoid text representation>.

   viii) If <curve text representation> immediately contains a <spiral text representation>, then <curve text representation> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral text representation>.

   ix) Otherwise, <curve text representation> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurve text representation>.

d) <linestring text representation> is the well-known text representation for an *ST_LineString* value. <linestring text representation> produces an *ST_LineString* value specified by the immediately contained <linestring text body>.

e) <circularstring text representation> is the well-known text representation for an *ST_CircularString* value. Let *APA* be the *ST_Point* ARRAY value produced by a <circularstring text>.

   Case:

   i) If the cardinality of *APA* is 0 (zero), then <circularstring text representation> produces an empty set of type *ST_CircularString*.

   ii) Otherwise, <circularstring text representation> produces an *ST_CircularString* value as the result of the value expression: *NEW ST_CircularString(APA)*.

f) <circle text representation> is the well-known text representation for an *ST_Circle* value. Let *APA* be the *ST_Point* ARRAY value produced by a <circle text>.

   Case:

   i) If the cardinality of *APA* is 0 (zero), then <circle text representation> produces an empty set of type *ST_Circle*.

   ii) Otherwise, <circle text representation> produces an *ST_Circle* value as the result of the value expression: *NEW ST_Circle(APA)*.

g) <geodesic text representation> is the well-known text representation for an *ST_GeodesicString* value. Let *APA* be the *ST_Point* ARRAY value produced by a <geodesic text>.

   Case:

i) If the cardinality of *APA* is 0 (zero), then <geodesic text representation> produces an empty set of type *ST_GeodesicString*.

ii) Otherwise, <geodesic text representation> produces an *ST_GeodesicString* value as the result of the value expression: *NEW ST_GeodesicString(APA)*.

h) <elliptical text representation> is the well-known text representation for an *ST_EllipticalCurve* value. <elliptical text representation> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical text>.

i) <nurbs text representation> is the well-known text representation for an *ST_NURBSCurve* value. <nurbs text representation> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs text>.

j) <clothoid text representation> is the well-known text representation for an *ST_Clothoid* value. <clothoid text representation> produces an *ST_Clothoid* value specified by the immediately contained <clothoid text>.

k) <spiral text representation> is the well-known text representation for an *ST_SpiralCurve* value. <spiral text representation> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral text>.

l) <compoundcurve text representation> is the well-known text representation for an *ST_CompoundCurve* value. Let *ACA* be the *ST_Curve* ARRAY value produced by a <compoundcurve text>.

Case:

i) If the cardinality of *ACA* is 0 (zero), then <compoundcurve text representation> produces an empty set of type *ST_CompoundCurve*.

ii) Otherwise, <compoundcurve text representation> produces an *ST_CompoundCurve* value as the result of the value expression: *NEW ST_CompoundCurve(ACA)*.

m) Case:

i) If <surface text representation> immediately contains a <curvepolygon text representation>, then <surface text representation> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygon text representation>.

ii) If <surface text representation> immediately contains a <polyhedralsurface text representation>, then <surface text representation> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurface text representation>.

iii) Otherwise, <surface text representation> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurface text representation>.

n) <curvepolygon text representation> is the well-known text representation for an *ST_CurvePolygon* value.

Case:

i) If <curvepolygon text representation> immediately contains a <curvepolygon text body>, then <curvepolygon text representation> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygon text body>.

ii) Otherwise, <curvepolygon text representation> produces an *ST_Polygon* value specified by the immediately contained <polygon text representation>.

o) Case:

i) If <polygon text representation> immediately contains a <polygon text body>, then <polygon text representation> produces an *ST_Polygon* value specified by the immediately contained <polygon text body>.

ii) Otherwise, <polygon text representation> produces an *ST_Triangle* value specified by the immediately contained <triangle text representation>.

p) <triangle text representation> is the well-known text representation for an *ST_Triangle* value. <triangle text representation> produces an *ST_Triangle* value specified by the immediately contained <triangle text body>.

q) <polyhedralsurface text representation> is the well-known text representation for an *ST_PolyhdrlSurface* value.

   Case:

   i) If <polyhedralsurface text representation> immediately contains a <polyhedralsurface text body>, then <polyhedralsurface text representation> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurface text body>.

   ii) Otherwise, <polyhedralsurface text representation> produces an *ST_TIN* value specified by the immediately contained <tin text body>.

r) <tin text representation> is the well-known text representation for an *ST_TIN* value. <tin text representation> produces an *ST_TIN* value specified by the immediately contained <tin text body>.

s) <compoundsurface text representation> is the well-known text representation for an *ST_CompoundSurface* value. <compoundsurface text representation> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurface text>.

t) <solid text representation> produces an *ST_BRepSolid* value specified by the immediately contained <brepsolid text representation>.

u) <brepsolid text representation> is the well-known text representation for an *ST_BRepSolid* value. <brepsolid text representation> produces an *ST_BRepSolid* value specified by the immediately contained <brepsolid text>.

v) Case:

   i) If <collection text representation> immediately contains a <multipoint text representation>, then <collection text representation> produces an *ST_MultiPoint* value specified by the immediately contained <multipoint text representation>.

   ii) If <collection text representation> immediately contains a <multicurve text representation>, then <collection text representation> produces an *ST_MultiCurve* value specified by the immediately contained <multicurve text representation>.

   iii) If <collection text representation> immediately contains a <multisurface text representation>, then <collection text representation> produces an *ST_MultiSurface* value specified by the immediately contained <multisurface text representation>.

   iv) Otherwise, <collection text representation> produces an *ST_GeomCollection* value specified by the immediately contained <geometrycollection text representation>.

w) <multipoint text representation> is the well-known text representation for an *ST_MultiPoint* value. Let *APA* be the *ST_Point* ARRAY value produced by a <multipoint text>.

   Case:

   i) If the cardinality of *APA* is 0 (zero), then <multipoint text representation> produces an empty set of type *ST_MultiPoint*.

   ii) Otherwise, <multipoint text representation> produces an *ST_MultiPoint* value as the result of the value expression: *NEW ST_MultiPoint(APA)*.

x) Case:

   i) If <multicurve text representation> immediately contains a <multicurve text>, then <multicurve text representation> produces an *ST_MultiCurve* value. Let *ACA* be the *ST_Curve* ARRAY value produced by a <multicurve text>.

      Case:

      1) If the cardinality of *ACA* is 0 (zero), then <multicurve text representation> produces an empty set of type *ST_MultiCurve*.

2) Otherwise, <multicurve text representation> produces an *ST_MultiCurve* value as the result of the value expression: *NEW ST_MultiCurve(ACA)*.

ii) Otherwise, <multicurve text representation> produces an *ST_MultiLineString* value specified by the immediately contained <multilinestring text representation>.

y) <multilinestring text representation> is the well-known text representation for an *ST_MultiLineString* value. Let *ALSA* be the *ST_LineString* ARRAY value produced by a <multilinestring text>.

Case:

i) If the cardinality of *ALSA* is 0 (zero), then <multilinestring text representation> produces an empty set of type *ST_MultiLineString*.

ii) Otherwise, <multilinestring text representation> produces an *ST_MultiLineString* value as the result of the value expression: *NEW ST_MultiLineString(ALSA)*.

z) Case:

i) If <multisurface text representation> immediately contains a <multisurface text>, then <multisurface text representation> produces an *ST_MultiSurface* value. Let *ASA* be the *ST_Surface* ARRAY value produced by a <multisurface text>.

Case:

1) If the cardinality of *ASA* is 0 (zero), then <multisurface text representation> produces an empty set of type *ST_MultiSurface*.

2) Otherwise, <multisurface text representation> produces an *ST_MultiSurface* value as the result of the value expression: *NEW ST_MultiSurface(ASA)*.

ii) Otherwise, <multisurface text representation> produces an *ST_MultiPolygon* value specified by the immediately contained <multipolygon text representation>.

aa) <multipolygon text representation> is the well-known text representation for an *ST_MultiPolygon* value. Let *APA* be the *ST_Polygon* ARRAY value produced by a <multipolygon text>.

Case:

i) If the cardinality of *APA* is 0 (zero), then <multipolygon text representation> produces an empty set of type *ST_MultiPolygon*.

ii) Otherwise, <multipolygon text representation> produces an *ST_MultiPolygon* value as the result of the value expression: *NEW ST_MultiPolygon(APA)*.

ab) <geometrycollection text representation> is the well-known text representation for an *ST_GeomCollection*. Let *AGA* be the *ST_Geometry* ARRAY value produced by a <geometrycollection text>.

Case:

i) If the cardinality of *AGA* is 0 (zero), then <geometrycollection text representation> produces an empty set of type *ST_GeomCollection*.

ii) Otherwise, <geometrycollection text representation> produces an *ST_GeomCollection* value as the result of the value expression: *NEW ST_GeomCollection(AGA)*.

ac) Let *APA* be the *ST_Point* ARRAY value produced by a <linestring text> in <linestring text body>.

Case:

i) If the cardinality of *APA* is 0 (zero), then <linestring text body> produces an empty set of type *ST_LineString*.

ii) Otherwise, <linestring text body> produces an *ST_LineString* value as the result of the value expression: *NEW ST_LineString(APA)*.

ad) Let *ACA* be the *ST_Curve* ARRAY value produced by a <curvepolygon text> in <curvepolygon text body>.

Case:

i) If the cardinality of *ACA* is 0 (zero), then <curvepolygon text body> produces an empty set of type *ST_CurvePolygon*.

ii) If the cardinality of *ACA* is 1 (one), then let *AER* be the element of *ACA*. <curvepolygon text body> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER)*.

iii) Otherwise, let *AER* be the first element in *ACA* and let *AIR* be the sublist of *ACA* containing the other elements of *ACA*. <curvepolygon text body>< produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER, AIR)*.

ae) Let *ALSA* be the *ST_LineString* ARRAY value produced by a <polygon text> in <polygon text body>.

Case:

i) If the cardinality of *ALSA* is 0 (zero), then <polygon text body> produces an empty set of type *ST_Polygon*.

ii) If the cardinality of *ALSA* is 1 (one), then let *ALS* be the element of *ALSA*. <polygon text body> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon*(*ALS*).

iii) Otherwise, let *AER* be the first element in *ALSA* and let *AIR* be the sublist of *ALSA* containing the other elements of *ALSA*. <polygon text body> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(AER, AIR)*.

af) Let *P1, P2 and P3* be the *ST_Point* values produced by a <triangle text> in <triangle text body>.

Case:

i) If *P1, P2, P3* values are not provided, then <polygon text body> produces an empty set of type *ST_Triangle*.

ii) Otherwise, let *APA* be an ST_Point ARRAY containing four elements: *P1, P2, P3, P1*. <triangle text body> produces an *ST_Triangle* value as the result of the value expression: *NEW ST_Triangle*(*APA*).

ag) Let *APA* be the *ST_Polygon* ARRAY value produced by a <polygonpatches text> in <polyhedralsurface text>.

Case:

i) If the cardinality of *APA* is 0 (zero), then <polyhedralsurface text> produces an empty set of type *ST_PolyhdrlSurface*.

ii) Otherwise, <polyhedralsurface text> produces an *ST_PolyhdrlSurface* value as the result of the value expression: *NEW ST_PolyhdrlSurface(APA)*.

ah) Let *ATA* be the *ST_Triangle* ARRAY value produced by the <trianglepatches text> and *AEA* be the ST_TINElement ARRAY value produced by the <tinelement list> and MSL be the DOUBLE PRECISION value produced by the <maxsidelength>, all in the same <tin text>.

Case:

i) If the cardinality of *ATA* is 0 (zero), then <tin text> produces an empty set of type *ST_TIN*.

ii) Otherwise, <tin text> produces an *ST_TIN* value as the result of the value expression: *NEW ST_TIN(ATA, AEA, MSL)*.

ai) Case:

i) If any <well-known text representation> *WKT1* contains <z m>, then every <well-known text representation> *WKT2* contained in *WKT1* shall contain <z m> with the same value.

ii) Otherwise, an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

aj) Case:

i) If any <well-known text representation> *WKT3* contains <just z>, then every <well-known text representation> *WKT4* contained in *WKT3* shall contain <just z> with the same value.

ii) Otherwise, an exception condition is raised: SQL/MM Spatial exception – mixed coordinate dimensions.

ak) Case:

i) If any <well-known text representation> *WKT5* contains both <z m> and <just z>, then for every <well-known text representation> *WKT6* contained in *WKT5:*

Case:

1) If <z m> has a value of ZM or Z, then <just z> shall have a value of Z.

2) Otherwise, <just z> shall not be included in *WKT6.*

ii) Otherwise, an exception condition is raised: SQL/MM Spatial exception – mixed coordinate dimensions.

al) Case:

i) If <z m> is specified, then

Case:

1) If <z m> immediately contains ZM, then let *ZORM* be ZM.

2) If <z m> immediately contains Z, then let *ZORM* be Z.

3) If <z m> immediately contains M, then let *ZORM* be M.

ii) Otherwise, let *ZORM* be 2D.

am) Case:

i) If <just z> is specified, then

Case:

1) If <just z> immediately contains Z, then let *ZORM* be Z.

ii) Otherwise, let *ZORM* be 2D.

an) Case:

i) If <point text> immediately contains an <empty set>, then:

Case:

1) If *ZORM* is ZM, then <point text> produces an empty set of type *ST_Point* as the result of the value expression: *NEW ST_Point(NULL, NULL, NULL, NULL)*.

2) If *ZORM* is Z, then <point text> produces an empty set of type *ST_Point* as the result of the value expression: *NEW ST_Point(NULL, NULL, NULL)*.

3) If *ZORM* is M, then <point text> produces an empty set of type *ST_Point* as the result of the value expression: *NEW ST_Point(NULL, NULL, NULL, 0)*.

4) Otherwise, <point text> produces an empty set of type *ST_Point* as the result of the value expression: *NEW ST_Point()*.

ii) Otherwise, <point text> produces the *ST_Point* value from <point>.

ao) Let *XC* be the DOUBLE PRECISION value specified by <x> in <point> and *YC* be the DOUBLE PRECISION value specified by <y> in <point>.

Case:

i) If *ZORM* is ZM then,

1) If <point> does not contain <z> or <point> does not contain <m>, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

2) Let *ZC* be the DOUBLE PRECISION value specified by <z> in <point>.

3) Let *MC* be the DOUBLE PRECISION value specified by <m> in <point>.

4) <point> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC, ZC, MC)*.

ii) If *ZORM* is Z then,

1) If <point> does not contain <z> or <point> contains <m>, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

2) Let *ZC* be the DOUBLE PRECISION value specified by <z> in <point>.

3) <point> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC, ZC)*.

iii) If *ZORM* is M then,

1) If <point> contains <z> or <point> does not contain <m>, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

2) Let *MC* be the DOUBLE PRECISION value specified by <m> in <point>.

3) <point> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC, NULL, MC)*.

iv) Otherwise,

1) If <point> contains <z> or <point> contains <m>, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

2) <point> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC)*.

ap) Case:

i) If <linestring text> immediately contains an <empty set>, then <linestring text> produces an empty *ST_Point* ARRAY value.

ii) Otherwise, <linestring text> produces an *ST_Point* ARRAY value that contains the *ST_Point* values specified by the immediately contained <point>s.

aq) Case:

i) If <circularstring text> immediately contains an <empty set>, then <circularstring text> produces an empty *ST_Point* ARRAY value.

ii) Otherwise, <circularstring text> produces an *ST_Point* ARRAY value that contains the *ST_Point* values specified by the immediately contained <point>s.

ar) Case:

i) If <circle text> immediately contains an <empty set>, then <circle text> produces an empty *ST_Point* ARRAY value.

ii) Otherwise, <circle text> produces an *ST_Point* ARRAY value that contains the *ST_Point* values specified by the immediately contained <point>s.

as) Case:

i) If <geodesic text> immediately contains an <empty set>, then <geodesic text> produces an empty *ST_Point* ARRAY value.

ii) Otherwise, <geodesic text> produces an *ST_Point* ARRAY value that contains the *ST_Point* values specified by the immediately contained <point>s.

at) Case:

i) If <elliptical text> immediately contains an <empty set>, then <elliptical text> produces an empty *ST_EllipticalCurve* value.

ii) Otherwise,

Case:

1) If *ZORM* is ZM or M, then

a) If <elliptical text> does not contain <startangle text representation> and <endangle text representation>, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s).*

b) <elliptical text> produces an *ST_EllipticalCurve* value from the immediately contained <referencelocation text representation>, <uaxislength text representation>, <vaxislength text representation>, <startangle text representation>, <endangle text representation>, <startm text representation> and <endm text representation>.

2) Otherwise, <elliptical text> produces an *ST_EllipticalCurve* value from the immediately contained <referencelocation text representation>, <uaxislength text representation>, <vaxislength text representation>, <startangle text representation> and <endangle text representation>.

au) <referencelocation text representation> is the well-known text representation for an *ST_EllipticalCurve* reference location attribute value. <referencelocation text representation> produces an *ST_EllipticalCurve* reference location attribute value specified by the immediately contained <affineplacement text representation>.

av) <uaxislength text representation> is the well-known text representation for an *ST_EllipticalCurve* u axis length attribute value. <uaxislength text representation> produces an *ST_EllipticalCurve* u axis length attribute value specified by the immediately contained <length text>.

aw) <vaxislength text representation> is the well-known text representation for an *ST_EllipticalCurve* v axis length attribute value. <vaxislength text representation> produces an *ST_EllipticalCurve* v axis length attribute value specified by the immediately contained <length text>.

ax) <startangle text representation> is the well-known text representation for an *ST_EllipticalCurve* start angle attribute value. <startangle text representation> produces an *ST_EllipticalCurve* start angle attribute value specified by the immediately contained <angle text>.

ay) <endangle text representation> is the well-known text representation for an *ST_EllipticalCurve* end angle attribute value. <endangle text representation> produces an *ST_EllipticalCurve* end angle attribute value specified by the immediately contained <angle text>.

az) <startm text representation> is the well-known text representation for a start m attribute value. <startm text representation> produces a start m attribute value specified by the immediately contained <number>.

ba) <endm text representation> is the well-known text representation for an end m attribute value. <endm text representation> produces an end m attribute value specified by the immediately contained <number>.

bb) Case:

i) If <length text> immediately contains an <empty set>, then <length text> produces an empty *ST_EllipticalCurve* u axis length or v axis length attribute value.

ii) Otherwise, <length text> produces an *ST_EllipticalCurve* u axis length or v axis length attribute value from the immediately contained <number>.

bc) <affineplacement text representation> is the well-known text representation for an *ST_AffinePlacement* value. <affineplacement text representation> produces an *ST_AffinePlacement* value specified by the immediately contained <affineplacement text>.

bd) Case:

i) If <affineplacement text> immediately contains an <empty set>, then <affineplacement text> produces an empty *ST_AffinePlacement* value.

ii) Otherwise, <affineplacement text> produces an *ST_AffinePlacement* value from the immediately contained <location text representation> and <referencedirections text representation>.

be) <location text representation> is the well-known text representation for an *ST_AffinePlacement* location attribute value. <location text representation> produces an *ST_AffinePlacement* location attribute value specified by the immediately contained <point text>.

bf) <referencedirections text representation> is the well-known text representation for an *ST_AffinePlacement* reference directions attribute value. <referencedirections text representation> produces an *ST_AffinePlacement* reference directions attribute value specified by the immediately contained <referencedirections text>.

bg) Case:

i) If <referencedirections text> immediately contains an <empty set>, then <referencedirections text> produces an empty *ST_AffinePlacement* reference directions attribute value.

ii) Otherwise, <referencedirections text> produces an *ST_AffinePlacement* reference directions attribute value from the immediately contained <vector text representation>'s.

bh) Case:

i) If <nurbs text> immediately contains an <empty set>, then <nurbs text> produces an empty *ST_NURBSCurve* value.

ii) Otherwise,

Case:

1) If *ZORM* is ZM or M, then

a) If <nurbs text> does not contain <startangle text representation> and <endangle text representation>, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

b) <nurbs text> produces an ST_NURBSCurve value from the immediately contained <degree text representation>, <controlpoints text representation>, <knots text representation>, <startm text representation> and <endm text representation>.

2) Otherwise, <nurbs text> produces an ST_NURBSCurve value from the immediately contained <degree text representation>, <controlpoints text representation>, and <knots text representation>.

bi) <degree text representation> is the well-known text representation for an *ST_NURBSCurve* degree attribute value. <degree text representation> produces an *ST_NURBSCurve* degree attribute value specified by the immediately contained <signed integer>.

bj) <controlpoints text representation> is the well-known text representation for an *ST_NURBSCurve* control points attribute value. <controlpoints text representation> produces an *ST_NURBSCurve* control points attribute value specified by the immediately contained <controlpoints text>.

bk) <knots text representation> is the well-known text representation for an *ST_NURBSCurve* knots attribute value. <knots text representation> produces an *ST_NURBSCurve* knots attribute value specified by the immediately contained <knots text>.

bl) Case:

i) If <controlpoints text> immediately contains an <empty set>, then <controlpoints text> produces an empty *ST_NURBSCurve* control points attribute value.

ii) Otherwise, <controlpoints text> produces an *ST_NURBSCurve* controlpoints attribute value from the immediately contained <nurbspoint text representation>s.

bm) <nurbspoint text representation> is the well-known text representation for an *ST_NURBSPoint value*. <nurbspoint text representation> produces an *ST_NURBSPoint value* specified by the immediately contained <nurbspoint text>.

bn) Case:

i) If <nurbspoint text> immediately contains an <empty set>, then <nurbspoint text> produces an empty *ST_NURBSPoint* value.

ii) Otherwise, <nurbspoint text> produces an *ST_NURBSPoint value* from the immediately contained <weightedpoint text representation> and <weight text representation>.

bo) <weightedpoint text representation> is the well-known text representation for an *ST_NURBSPoint* weighted point attribute value. <weightedpoint text representation> produces an *ST_NURBSPoint* weighted point attribute value specified by the immediately contained <point text>.

bp) <weight text representation> is the well-known text representation for an *ST_NURBSPoint* weight attribute value. <weight text representation> produces an *ST_NURBSPoint* weight attribute value specified by the immediately contained <number>.

bq) Case:

i) If <knots text> immediately contains an <empty set>, then <knots text> produces an empty *ST_NURBSCurve* knots attribute value.

ii) Otherwise, <knots text> produces an *ST_NURBSCurve* knots attribute value from the immediately contained <knot text representation>s.

br) <knot text representation> is the well-known text representation for an *ST_Knot* value. <knot text representation> produces an *ST_Knot* value specified by the immediately contained <knot text>.

bs) Case:

i) If <knot text> immediately contains an <empty set>, then <knot text> produces an empty *ST_Knot* value.

ii) Otherwise, <knot text> produces an *ST_Knot* value from the immediately contained <value text representation> and <multiplicity text representation>.

bt) <value text representation> is the well-known text representation for an *ST_Knot* value attribute value. <value text representation> produces an *ST_Knot* value attribute value specified by the immediately contained <number>.

bu) <multiplicity text representation> is the well-known text representation for an *ST_Knot* multiplicity attribute value. <multiplicity text representation> produces an *ST_Knot* multiplicity attribute value specified by the immediately contained <signed integer>.

bv) Case:

i) If <clothoid text> immediately contains an <empty set>, then <clothoid text> produces an empty *ST_Clothoid* value.

ii) Otherwise,

Case:

1) If *ZORM* is ZM or M, then

a) If <clothoid text> does not contain <startangle text representation> and <endangle text representation>, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s).*

b) <clothoid text> produces an *ST_Clothoid* value from the immediately contained <referencelocation text representation>, <scalefactor text representation>, <startdistance text representation>, <enddistance text representation>, <startm text representation> and <endm text representation>.

2) Otherwise, <clothoid text> produces an ST_Clothoid value from the immediately contained <referencelocation text representation>, <scalefactor text representation>, <startdistance text representation>, and <enddistance text representation>.

bw) <scalefactor text representation> is the well-known text representation for an *ST_Clothoid* scale factor attribute value. <scalefactor text representation> produces an *ST_Clothoid* scale factor attribute value specified by the immediately contained <scalefactor text>.

bx) <startdistance text representation> is the well-known text representation for an *ST_Clothoid* start distance attribute value. <startdistance text representation> produces an *ST_Clothoid* start distance attribute value specified by the immediately contained <distance text>.

by) <enddistance text representation> is the well-known text representation for an *ST_Clothoid* end distance attribute value. <enddistance text representation> produces an *ST_Clothoid* end distance attribute value specified by the immediately contained <distance text>.

bz) Case:

   i) If <scalefactor text> immediately contains an <empty set>, then <scalefactor text> produces an empty *ST_Clothoid* scale factor attribute value.

   ii) Otherwise, <scalefactor text> produces an *ST_Clothoid* scale factor attribute value from the immediately contained <number>.

ca) Case:

   i) If <distance text> immediately contains an <empty set>, then <distance text> produces an empty *ST_Clothoid* start or end distance attribute value.

   ii) Otherwise, <distance text> produces an *ST_Clothoid* start or end distance attribute value from the immediately contained <number>.

cb) Case:

   i) If <spiral text> immediately contains an <empty set>, then <spiral text> produces an empty *ST_SpiralCurve* value.

   ii) Otherwise,

   Case:

      1) If *ZORM* is ZM or M, then

         a) If <spiral text> does not contain <startangle text representation> and <endangle text representation>, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

         b) <spiral text> produces an *ST_EllipticalCurve* value from the immediately contained <referencelocation text representation>, <spirallength text representation>, <startcurvature text representation>, <endcurvature text representation>, <spiraltype text representation>, <startm text representation> and <endm text representation>.

      2) Otherwise, <spiral text> produces an ST_EllipticalCurve value from the immediately contained <referencelocation text representation>, <spirallength text representation>, <startcurvature text representation>, <endcurvature text representation> and <spiraltype text representation>.

cc) <spirallength text representation> is the well-known text representation for an *ST_SpiralCurve* length attribute value. <spirallength text representation> produces an *ST_SpiralCurve* length attribute value specified by the immediately contained <spirallength text>.

cd) <startcurvature text representation> is the well-known text representation for an *ST_SpiralCurve* start curvature attribute value. <startcurvature text representation> produces an *ST_SpiralCurve* start curvature attribute value specified by the immediately contained <curvature text>.

ce) <endcurvature text representation> is the well-known text representation for an *ST_SpiralCurve* end curvature attribute value. <endcurvature text representation> produces an *ST_SpiralCurve* end curvature attribute value specified by the immediately contained <curvature text>.

cf) <spiraltype text representation> is the well-known text representation for an *ST_SpiralCurve* spiral type attribute value. <spiraltype text representation> produces an *ST_SpiralCurve* spiral type attribute value specified by the immediately contained <spiraltype text>.

cg) Case:

   i) If <spirallength text> immediately contains an <empty set>, then <spirallength text> produces an empty *ST_SpiralCurve* spiral length attribute value.

   ii) Otherwise, <spirallength text> produces an *ST_SpiralCurve* spiral length attribute value from the immediately contained <number>.

ch) Case:

i) If <curvature text> immediately contains an <empty set>, then <curvature text> produces an empty *ST_SpiralCurve* start or end curvature attribute value.

ii) Otherwise, <curvature text> produces an *ST_SpiralCurve* start or end curvature attribute value from the immediately contained <number>.

ci) Case:

i) If <spiraltype text> immediately contains an <empty set>, then <spiraltype text> produces an empty *ST_SpiralCurve* spiral type attribute value.

ii) Otherwise, <spiraltype text> produces an *ST_SpiralCurve* spiral type attribute value from the immediately contained <letters>.

cj) Case:

i) If <compoundcurve text> immediately contains an <empty set>, then <compoundcurve text> produces an empty *ST_Curve* ARRAY value.

ii) Otherwise, <compoundcurve text> produces an *ST_Curve* ARRAY value that contains the *ST_Curve* values specified by the immediately contained <curve text>s.

ck) Case:

i) If <curve text> immediately contains a <linestring text body>, then <curve text> produces an *ST_LineString* value specified by the immediately contained <linestring text body>.

ii) If <curve text> immediately contains a <circularstring text representation>, then <curve text> produces an *ST_CircularString* value specified by the immediately contained <circularstring text representation>.

iii) If <curve text> immediately contains a <circle text representation>, then <curve text> produces an *ST_Circle* value specified by the immediately contained <circle text representation>.

iv) If <curve text> immediately contains a <geodesic text representation>, then <curve text> produces an *ST_GeodesicString* value specified by the immediately contained <geodesic text representation>.

v) If <curve text> immediately contains an <elliptical text representation>, then <curve text> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical text representation>.

vi) If <curve text> immediately contains a <nurbs text representation>, then <curve text> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs text representation>.

vii) If <curve text> immediately contains a <clothoid text representation>, then <curve text> produces an *ST_Clothoid* value specified by the immediately contained <clothoid text representation>.

viii) If <curve text> immediately contains a <spiral text representation>, then <curve text> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral text representation>.

ix) Otherwise, <curve text> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurve text representation>.

cl) Case:

i) If <ring text> immediately contains a <linestring text body>, then <ring text> produces an *ST_LineString* value specified by the immediately contained <linestring text body>.

ii) If <ring text> immediately contains a <circularstring text representation>, then <ring text> produces an *ST_CircularString* value specified by the immediately contained <circularstring text representation>.

iii) If <ring text> immediately contains a <circle text representation>, then <ring text> produces an *ST_Circle* value specified by the immediately contained <circle text representation>.

iv) If <ring text> immediately contains a <geodesic text representation>, then <ring text> produces an *ST_GeodesicString* value specified by the immediately contained <geodesic text representation>.

v) If <ring text> immediately contains a <elliptical text representation>, then <ring text> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical text representation>.

vi) If <ring text> immediately contains a <nurbs text representation>, then <ring text> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs text representation>.

vii) If <ring text> immediately contains a <clothoid text representation>, then <ring text> produces an *ST_Clothoid* value specified by the immediately contained <clothoid text representation>.

viii) If <ring text> immediately contains a <spiral text representation>, then <ring text> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral text representation>.

ix) Otherwise, <ring text> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurve text representation>.

cm) Case:

i) If <surface text> immediately contains a <curvepolygon text body>, then <surface text> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygon text body>.

ii) If <surface text> immediately contains a <polygon text body>, then <surface text> produces an *ST_Polygon* value specified by the immediately contained <polygon text body>.

iii) If <surface text> immediately contains a <triangle text body>, then <surface text> produces an *ST_Triangle* value specified by the immediately contained <triangle text body>.

iv) If <surface text> immediately contains a <polyhedralsurface text body>, then <surface text> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurface text body>.

v) If <surface text> immediately contains a <tin text body>, then <surface text> produces an *ST_TIN* value specified by the immediately contained <tin text body>.

vi) Otherwise, <surface text> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurface text body>.

cn) Case:

i) If <curvepolygon text> immediately contains an <empty set>, then <curvepolygon text> produces an empty *ST_Curve* ARRAY value.

ii) Otherwise, <curvepolygon text> produces an *ST_Curve* ARRAY value that contains the *ST_Curve* values specified by the immediately contained <ring text>s.

co) Case:

i) If <polygon text> immediately contains an <empty set>, then <polygon text> produces an empty *ST_LineString* ARRAY value.

ii) Otherwise, <polygon text> produces an *ST_LineString* ARRAY value that contains the *ST_LineString* values specified by the immediately contained <linestring text>s.

cp) Case:

i) If <triangle text> immediately contains an <empty set>, then <triangle text> produces an empty *ST_Point* ARRAY value.

ii) Otherwise, <triangle text> produces an *ST_Point* ARRAY value that contains the three *ST_Point* values specified by the immediately contained <point>s.

cq) Case:

i) If <polyhedralsurface text> immediately contains an <empty set>, then <polyhedralsurface text> produces an empty *ST_Polygon* ARRAY value.

ii) Otherwise, <polyhedralsurface text> produces an *ST_Polygon* ARRAY value that contains the *ST_Polygon* values specified by the immediately contained <polygonpatches text>s.

cr) Case:

i) If <tin text> immediately contains an <empty set>, then <tin text> produces an empty *ST_Tin* value.

ii) Otherwise, <tin text> produces an *ST_Tin* value specified by the immediately contained <trianglepatches text>, <tinelement list>, and <maxsidelength>.

cs) Case:

i) If <compoundsurface text> immediately contains an <empty set>, then <compoundsurface text> produces an empty *ST_CompoundSurface* value.

ii) Otherwise, <compoundsurface text> produces an *ST_CompoundSurface* value that contains the *ST_Surface* values specified by the immediately contained <surface text>s.

ct) Case:

i) If <brepsolid text> immediately contains an <empty set>, then <brepsolid text> produces an empty *ST_BRepSolid* value.

ii) Otherwise, <brepsolid text> produces an *ST_BRepSolid* value that contains the *ST_Surface* values specified by the immediately contained <shell text>s.

cu) Case:

i) If <shell text> immediately contains a <polyhedralsurface text body>, then <shell text> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurface text body>.

ii) Otherwise, <shell text> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurface text representation>.

cv) Case:

i) If <multipoint text> immediately contains an <empty set>, then <multipoint text> produces an empty *ST_Point* ARRAY value.

ii) Otherwise, <multipoint text> produces an *ST_Point* ARRAY value that contains the *ST_Point* values specified by the immediately contained <point text>s.

cw) Case:

i) If <multicurve text> immediately contains an <empty set>, then <multicurve text> produces an empty *ST_Curve* ARRAY value.

ii) Otherwise, <multicurve text> produces an *ST_Curve* ARRAY value that contains the *ST_Curve* values specified by the immediately contained <curve text>s.

cx) Case:

i) If <multilinestring text> immediately contains an <empty set>, then <multilinestring text> produces an empty *ST_LineString* ARRAY value.

ii) Otherwise, <multilinestring text> produces an *ST_LineString* ARRAY value that contains the ST_LineString values specified by the immediately contained <linestring text body>s.

cy) Case:

i) If <multisurface text> immediately contains an <empty set>, then <multisurface text> produces an empty *ST_Surface* ARRAY value.

ii) Otherwise, <multisurface text> produces an *ST_Surface* ARRAY value that contains the ST_Surface values from the immediately contained <surface text>s.

cz) Case:

i) If <multipolygon text> immediately contains an <empty set>, then <multipolygon text> produces an empty *ST_Polygon* ARRAY value.

ii) Otherwise, <multipolygon text> produces an *ST_Polygon* ARRAY value that contains the ST_Polygon values from the immediately contained <polygon text body>s.

da) Case:

i) If <geometrycollection text> immediately contains an <empty set>, then <geometrycollection text> produces an empty *ST_Geometry* ARRAY value.

ii) Otherwise, an ST_Geometry ARRAY value that contains the ST_Geometry values from the immediately contained <well-known text representation>s.

db) The list of keywords is:

AFFINEPLACEMENT

BOUNDARY
BREAKLINE
BREAKVOID
BREPSOLID

CIRCLE
CIRCULARSTRING
CLOTHOID
COMPOUNDCURVE
COMPOUNDSURFACE
CONTROLCONTOUR
CONTROLPOINTS
CURVEPOLYGON

DEGREE
DRAPEVOID

ELEMENTS
ELLIPTICALCURVE
EMPTY
ENDANGLE
ENDCURVATURE
ENDDISTANCE
ENDM

GEODESICSTRING
GEOMETRYCOLLECTION
GROUPSPOT

HOLE

ID

KNOT
KNOTS

LENGTH
LINESTRING
LOCATION

M
MAXSIDELENGTH
MULTICURVE

MULTILINESTRING
MULTIPLICITY
MULTIPOINT
MULTIPOLYGON
MULTISURFACE

NURBSCURVE
NURBSPOINT

POINT
POINTS
POLYGON
POLYHEDRALSURFACE

REFERENCEDIRECTIONS
REFERENCELOCATION

SCALEFACTOR
SOFTBREAK
SPIRALCURVE
SPIRALTYPE
STARTANGLE
STARTCURVATURE
STARTDISTANCE
STARTM
STOPLINE

TAG
TIN
TRIANGLE

UAXISLENGTH

VALUE
VAXISLENGTH
VOID

WEIGHT
WEIGHTEDPOINT

Z
ZM

5.1.68 <well-known binary representation>

Purpose

This subclause contains the definition of <well-known binary representation>.

Description

1) The well-known binary representation of an *ST_Geometry* value is defined by the following BNF for <well-known binary representation>.

```
<well-known binary representation> ::=
     <well-knownzm binary representation>
   | <well-knownz binary representation>
   | <well-knownm binary representation>
   | <well-known2d binary representation>

<well-knownzm binary representation> ::=
     <pointzm binary representation>
   | <curvezm binary representation>
   | <surfacezm binary representation>
   | <collectionzm binary representation>

<well-knownz binary representation> ::=
     <pointz binary representation>
   | <curvez binary representation>
   | <surfacez binary representation>
   | <solidz binary representation>
   | <collectionz binary representation>

<well-knownm binary representation> ::=
     <pointm binary representation>
   | <curvem binary representation>
   | <surfacem binary representation>
   | <collectionm binary representation>

<well-known2d binary representation> ::=
     <point binary representation>
   | <curve binary representation>
   | <surface binary representation>
   | <collection binary representation>

<pointzm binary representation> ::=
     <byte order> <wkbpointzm> [ <wkbpointzm binary> ]

<pointz binary representation> ::=
     <byte order> <wkbpointz> [ <wkbpointz binary> ]

<pointm binary representation> ::=
     <byte order> <wkbpointm> [ <wkbpointm binary> ]

<point binary representation> ::=
     <byte order> <wkbpoint> [ <wkbpoint binary> ]

<curvezm binary representation> ::=
     <linestringzm binary representation>
   | <circularstringzm binary representation>
   | <circlezm binary representation>
   | <geodesiczm binary representation>
   | <ellipticalzm binary representation>
   | <nurbszm binary representation>
   | <clothoidzm binary representation>
   | <spiralzm binary representation>
   | <compoundcurvezm binary representation>

<curvez binary representation> ::=
     <linestringz binary representation>
```

```
    | <circularstringz binary representation>
    | <circlez binary representation>
    | <geodesicz binary representation>
    | <ellipticalz binary representation>
    | <nurbsz binary representation>
    | <clothoidz binary representation>
    | <spiralz binary representation>
    | <compoundcurvez binary representation>

<curvem binary representation> ::=
      <linestringm binary representation>
    | <circularstringm binary representation>
    | <circlem binary representation>
    | <geodesicm binary representation>
    | <ellipticalm binary representation>
    | <nurbsm binary representation>
    | <clothoidm binary representation>
    | <spiralm binary representation>
    | <compoundcurvem binary representation>

<curve binary representation> ::=
      <linestring binary representation>
    | <circularstring binary representation>
    | <circle binary representation>
    | <geodesic binary representation>
    | <elliptical binary representation>
    | <nurbs binary representation>
    | <clothoid binary representation>
    | <spiral binary representation>
    | <compoundcurve binary representation>

<linestringzm binary representation> ::=
      <byte order> <wkblinestringzm> [ <num> <wkbpointzm binary>... ]

<linestringz binary representation> ::=
      <byte order> <wkblinestringz> [ <num> <wkbpointz binary>... ]

<linestringm binary representation> ::=
      <byte order> <wkblinestringm> [ <num> <wkbpointm binary>... ]

<linestring binary representation> ::=
      <byte order> <wkblinestring> [ <num> <wkbpoint binary>... ]

<circularstringzm binary representation> ::=
      <byte order> <wkbcircularstringzm> [ <num> <wkbpointzm binary>... ]

<circularstringz binary representation> ::=
      <byte order> <wkbcircularstringz> [ <num> <wkbpointz binary>... ]

<circularstringm binary representation> ::=
      <byte order> <wkbcircularstringm> [ <num> <wkbpointm binary>... ]

<circularstring binary representation> ::=
      <byte order> <wkbcircularstring> [ <num> <wkbpoint binary>... ]

<circlezm binary representation> ::=
      <byte order> <wkbcirclezm> [ <num> <wkbpointzm binary>... ]

<circlez binary representation> ::=
      <byte order> <wkbcirclez> [ <num> <wkbpointz binary>... ]

<circlem binary representation> ::=
      <byte order> <wkbcirclem> [ <num> <wkbpointm binary>... ]

<circle binary representation> ::=
      <byte order> <wkbcircle> [ <num> <wkbpoint binary>... ]

<geodesiczm binary representation> ::=
```

```
     <byte order> <wkbgeodesiczm> [ <num> <wkbpointzm binary>... ]

<geodesicz binary representation> ::=
     <byte order> <wkbgeodesicz> [ <num> <wkbpointz binary>... ]

<geodesicm binary representation> ::=
     <byte order> <wkbgeodesicm> [ <num> <wkbpointm binary>... ]

<geodesic binary representation> ::=
     <byte order> <wkbgeodesic> [ <num> <wkbpoint binary>... ]

<ellipticalzm binary representation> ::=
     <byte order> <wkbellipticalzm> [ <wkbreferencelocationzm binary>
        <wkbuaxislength> <wkbvaxislength>
        <wkbstartangle> <wkbendangle>
        <wkbstartm> <wkbendm> ]

<ellipticalz binary representation> ::=
     <byte order> <wkbellipticalz> [ <wkbreferencelocationz binary>
        <wkbuaxislength> <wkbvaxislength> <wkbstartangle> <wkbendangle> ]

<ellipticalm binary representation> ::=
     <byte order> <wkbellipticalm> [ <wkbreferencelocationm binary>
        <wkbuaxislength> <wkbvaxislength>
        <wkbstartangle> <wkbendangle>
        <wkbstartm> <wkbendm> ]

<elliptical binary representation> ::=
     <byte order> <wkbelliptical> [ <wkbreferencelocation binary>
        <wkbuaxislength> <wkbvaxislength> <wkbstartangle> <wkbendangle> ]

<wkbreferencelocationzm binary> ::=
     <affineplacementz binary representation>

<wkbreferencelocationz binary> ::=
     <affineplacementz binary representation>

<wkbreferencelocationm binary> ::=
     <affineplacement binary representation>

<wkbreferencelocation binary> ::=
     <affineplacement binary representation>

<affineplacementz binary representation> ::=
     <byte order> <wkbaffineplacementz> [ <wkblocationz>
        <wkbreferencedirectionsz> ]

<affineplacement binary representation> ::=
     <byte order> <wkbaffineplacement> [ <wkblocation>
        <wkbreferencedirections> ]

<wkblocationz> ::=
     <wkbpointz binary>

<wkblocation> ::=
     <wkbpoint binary>

<wkbreferencedirectionsz> ::=
     <num> <wkbvectorz binary>...

<wkbreferencedirections> ::=
     <num> <wkbvector binary>...

<nurbszm binary representation> ::=
     <byte order> <wkbnurbszm> [ <wkbdegree> <wkbcontrolpointsz binary>
        <wkbknots binary>
        <wkbstartm> <wkbendm>]

<nurbsz binary representation> ::=
     <byte order> <wkbnurbszm> [ <wkbdegree> <wkbcontrolpointsz binary>
```

```
        <wkbknots binary> ]

<nurbsm binary representation> ::=
     <byte order> <wkbnurbszm> [ <wkbdegree> <wkbcontrolpoints binary>
        <wkbknots binary>
        <wkbstartm> <wkbendm> ]

<nurbs binary representation> ::=
     <byte order> <wkbnurbszm> [ <wkbdegree> <wkbcontrolpoints binary>
        <wkbknots binary> ]

<wkbcontrolpointsz binary> ::=
     <num> <nurbspointz binary representation>...

<wkbcontrolpoints binary> ::=
     <num> <nurbspoint binary representation>...

<nurbspointz binary representation> ::=
     <byte order> [ <wkbweightedpointz> <bit> [ <wkbweight> ] ]

<nurbspoint binary representation> ::=
     <byte order> [ <wkbweightedpoint> <bit> [ <wkbweight> ] ]

<wkbweightedpointz> ::=
     <wkbpointz binary>

<wkbweightedpoint> ::=
     <wkbpoint binary>

<wkbknots binary> ::=
     <num> <knot binary representation>...

<knot binary representation> ::=
     <byte order> [ <wkbvalue> <wkbmultiplicity> ]

<clothoidzm binary representation> ::=
     <byte order> <wkbclothoidzm> [ <wkbreferencelocationzm binary>
        <wkbscalefactor>
        <wkbstartdistance> <wkbenddistance>
        <wkbstartm> <wkbendm> ]

<clothoidz binary representation> ::=
     <byte order> <wkbclothoidz> [ <wkbreferencelocationz binary>
        <wkbscalefactor> <wkbstartdistance> <wkbenddistance> ]

<clothoidm binary representation> ::=
     <byte order> <wkbclothoidm> [ <wkbreferencelocationm binary>
        <wkbscalefactor>
        <wkbstartdistance> <wkbenddistance>
        <wkbstartm> <wkbendm> ]

<clothoid binary representation> ::=
     <byte order> <wkbclothoid> [ <wkbreferencelocation binary>
        <wkbscalefactor>
        <wkbstartdistance> <wkbenddistance> ]

<spiralzm binary representation> ::=
     <byte order> <wkbspiralzm> [ <wkbreferencelocationzm binary>
        <wkbspirallength>
        <wkbstartcurvature> <wkbendcurvature>
        <wkbspiraltype>
        <wkbstartm> <wkbendm> ]

<spiralz binary representation> ::=
     <byte order> <wkbspiralz> [ <wkbreferencelocationz binary>
        <wkbspirallength>
        <wkbstartcurvature> <wkbendcurvature>
        <wkbspiraltype> ]
```



```
<spiralm binary representation> ::=
     <byte order> <wkbspiralm> [ <wkbreferencelocationm binary>
        <wkbspirallength>
        <wkbstartcurvature> <wkbendcurvature>
        <wkbspiraltype>
        <wkbstartm> <wkbendm> ]

<spiral binary representation> ::=
     <byte order> <wkbspiral> [ <wkbreferencelocation binary>
        <wkbspirallength>
        <wkbstartcurvature> <wkbendcurvature>
        <wkbspiraltype> ]

<compoundcurvezm binary representation> ::=
     <byte order> <wkbcompoundcurvezm> [ <num> <wkbcurvezm binary>... ]

<compoundcurvez binary representation> ::=
     <byte order> <wkbcompoundcurvez> [ <num> <wkbcurvez binary>... ]

<compoundcurvem binary representation> ::=
     <byte order> <wkbcompoundcurvem> [ <num> <wkbcurvem binary>... ]

<compoundcurve binary representation> ::=
     <byte order> <wkbcompoundcurve> [ <num> <wkbcurve binary>... ]

<surfacezm binary representation> ::=
     <curvepolygonzm binary representation>
   | <polyhedralsurfacezm binary representation>
   | <compoundsurfacezm binary representation>

<surfacez binary representation> ::=
     <curvepolygonz binary representation>
   | <polyhedralsurfacez binary representation>
   | <compoundsurfacez binary representation>

<surfacem binary representation> ::=
     <curvepolygonm binary representation>
   | <polyhedralsurfacem binary representation>
   | <compoundsurfacem binary representation>

<surface binary representation> ::=
     <curvepolygon binary representation>
   | <polyhedralsurface binary representation>
   | <compoundsurface binary representation>

<solidz binary representation> ::=
     <brepsolidz binary representation>

<curvepolygonzm binary representation> ::=
     <byte order> <wkbcurvepolygonzm> [ <num> <wkbringzm binary>... ]
   | <polygonzm binary representation>

<curvepolygonz binary representation> ::=
     <byte order> <wkbcurvepolygonz> [ <num> <wkbringz binary>... ]
   | <polygonz binary representation>

<curvepolygonm binary representation> ::=
     <byte order> <wkbcurvepolygonm> [ <num> <wkbringm binary>... ]
   | <polygonm binary representation>

<curvepolygon binary representation> ::=
     <byte order> <wkbcurvepolygon> [ <num> <wkbring binary>... ]
   | <polygon binary representation>

<polygonzm binary representation> ::=
     <byte order> <wkbpolygonzm> [ <num> <wkblinearringzm binary>... ]
   | <trianglezm binary representation>

<polygonz binary representation> ::=
```

```
     <byte order> <wkbpolygonz> [ <num> <wkblinearringz binary>... ]
   | <trianglez binary representation>

<polygonm binary representation> ::=
     <byte order> <wkbpolygonm> [ <num> <wkblinearringm binary>... ]
   | <trianglem binary representation>

<polygon binary representation> ::=
     <byte order> <wkbpolygon> [ <num> <wkblinearring binary>... ]
   | <triangle binary representation>

<trianglezm binary representation> ::=
     <byte order> <wkbtrianglezm>
       [ <wkbpointzm binary> <wkbpointzm binary> <wkbpointzm binary> ]

<trianglez binary representation> ::=
     <byte order> <wkbtrianglez>
       [ <wkbpointz binary> <wkbpointz binary> <wkbpointz binary> ]

<trianglem binary representation> ::=
     <byte order> <wkbtrianglem>
       [ <wkbpointm binary> <wkbpointm binary> <wkbpointm binary> ]

<triangle binary representation> ::=
     <byte order> <wkbtriangle>
       [ <wkbpoint binary> <wkbpoint binary> <wkbpoint binary> ]

<polyhedralsurfacezm binary representation> ::=
     <byte order> <wkbpolyhedralsurfacezm>
       [ <num> <wkbpolygonpatchzm binary>... ]
   | <tinzm binary representation>

<polyhedralsurfacez binary representation> ::=
     <byte order> <wkbpolyhedralsurfacez>
       [ <num> <wkbpolygonpatchz binary>... ]
   | <tinz binary representation>

<polyhedralsurfacem binary representation> ::=
     <byte order> <wkbpolyhedralsurfacem>
       [ <num> <wkbpolygonpatchm binary>... ]
   | <tinm binary representation>

<polyhedralsurface binary representation> ::=
     <byte order> <wkbpolyhedralsurface>
       [ <num> <wkbpolygonpatch binary>... ]
   | <tin binary representation>

<tinzm binary representation> ::=
     <byte order> <wkbtinzm>
       [ <num> <wkbtrianglepatchzm binary>...
         <nume> <wkbtinelement binary>...
         <wkbmaxsidelength> ]

<tinz binary representation> ::=
     <byte order> <wkbtinz>
       [ <num> <wkbtrianglepatchz binary>...
         <nume> <wkbtinelement binary>...
         <wkbmaxsidelength> ]

<tinm binary representation> ::=
     <byte order> <wkbtinm>
       [ <num> <wkbtrianglepatchm binary>...
         <nume> <wkbtinelement binary>...
         <wkbmaxsidelength> ]

<tin binary representation> ::=
     <byte order> <wkbtin>
       [ <num> <wkbtrianglepatch binary>...
```

```
          <nume> <wkbtinelement binary>...
          <wkbmaxsidelength> ]

<tinelement binary representation> ::=
   <byte order> <tinelement element type> <tinelement element id>
   <tinelement element tag> <well-known binary representation>

<tinelement element type> ::=
     <byte> <letters>

<tinelement element id> ::=
     <signed integer>

<tinelement element tag> ::=
     <byte> <letters>

<compoundsurfacezm binary representation> ::=
     <byte order> <wkbcompoundsurfacezm>
       [ <num> <surfacezm binary representation>... ]

<compoundsurfacez binary representation> ::=
     <byte order> <wkbcompoundsurfacez>
       [ <num> <surfacez binary representation>... ]

<compoundsurfacem binary representation> ::=
     <byte order> <wkbcompoundsurfacem>
       [ <num> <surfacem binary representation>... ]

<compoundsurface binary representation> ::=
     <byte order> <wkbcompoundsurface>
       [ <num> <surface binary representation>... ]

<brepsolidz binary representation> ::=
     <byte order> <wkbbrepsolidz> [ <num> <wkbshellz binary>... ]

<collectionzm binary representation> ::=
     <multipointzm binary representation>
   | <multicurvezm binary representation>
   | <multisurfacezm binary representation>
   | <geometrycollectionzm binary representation>

<collectionz binary representation> ::=
     <multipointz binary representation>
   | <multicurvez binary representation>
   | <multisurfacez binary representation>
   | <geometrycollectionz binary representation>

<collectionm binary representation> ::=
     <multipointm binary representation>
   | <multicurvem binary representation>
   | <multisurfacem binary representation>
   | <geometrycollectionm binary representation>

<collection binary representation> ::=
     <multipoint binary representation>
   | <multicurve binary representation>
   | <multisurface binary representation>
   | <geometrycollection binary representation>

<multipointzm binary representation> ::=
     <byte order> <wkbmultipointzm>
        [ <num> <pointzm binary representation>... ]

<multipointz binary representation> ::=
     <byte order> <wkbmultipointz>
        [ <num> <pointz binary representation>... ]

<multipointm binary representation> ::=
     <byte order> <wkbmultipointm>
```

```
        [ <num> <pointm binary representation>... ]

<multipoint binary representation> ::=
     <byte order> <wkbmultipoint>
        [ <num> <point binary representation>... ]

<multicurvezm binary representation> ::=
     <byte order> <wkbmulticurvezm>
        [ <num> <curvezm binary representation>... ]
   | <multilinestringzm binary representation>

<multicurvez binary representation> ::=
     <byte order> <wkbmulticurvez>
        [ <num> <curvez binary representation>... ]
   | <multilinestringz binary representation>

<multicurvem binary representation> ::=
     <byte order> <wkbmulticurvem>
        [ <num> <curvem binary representation>... ]
   | <multilinestringm binary representation>

<multicurve binary representation> ::=
     <byte order> <wkbmulticurve>
        [ <num> <curve binary representation>... ]
   | <multilinestring binary representation>

<multilinestringzm binary representation> ::=
     <byte order> <wkbmultilinestringzm>
        [ <num> <linestringzm binary representation>... ]

<multilinestringz binary representation> ::=
     <byte order> <wkbmultilinestringz>
        [ <num> <linestringz binary representation>... ]

<multilinestringm binary representation> ::=
     <byte order> <wkbmultilinestringm>
        [ <num> <linestringm binary representation>... ]

<multilinestring binary representation> ::=
     <byte order> <wkbmultilinestring>
        [ <num> <linestring binary representation>... ]

<multisurfacezm binary representation> ::=
     <byte order> <wkbmultisurfacezm>
        [ <num> <surfacezm binary representation>... ]
   | <multipolygonzm binary representation>

<multisurfacez binary representation> ::=
     <byte order> <wkbmultisurfacez>
        [ <num> <surfacez binary representation>... ]
   | <multipolygonz binary representation>

<multisurfacem binary representation> ::=
     <byte order> <wkbmultisurfacem>
        [ <num> <surfacem binary representation>... ]
   | <multipolygonm binary representation>

<multisurface binary representation> ::=
     <byte order> <wkbmultisurface>
        [ <num> <surface binary representation>... ]
   | <multipolygon binary representation>

<multipolygonzm binary representation> ::=
     <byte order> <wkbmultipolygonzm>
        [ <num> <polygonzm binary representation>... ]

<multipolygonz binary representation> ::=
     <byte order> <wkbmultipolygonz>
```

```
        [ <num> <polygonz binary representation>... ]

<multipolygonm binary representation> ::=
     <byte order> <wkbmultipolygonm>
        [ <num> <polygonm binary representation>... ]

<multipolygon binary representation> ::=
     <byte order> <wkbmultipolygon>
        [ <num> <polygon binary representation>... ]

<geometrycollectionzm binary representation> ::=
     <byte order> <wkbgeometrycollectionzm>
        [ <num> <well-knownzm binary representation>... ]

<geometrycollectionz binary representation> ::=
     <byte order> <wkbgeometrycollectionz>
        [ <num> <well-knownz binary representation>... ]

<geometrycollectionm binary representation> ::=
     <byte order> <wkbgeometrycollectionm>
        [ <num> <well-knownm binary representation>... ]

<geometrycollection binary representation> ::=
     <byte order> <wkbgeometrycollection>
        [ <num> <well-known binary representation>... ]

<wkbpolygonpatchzm binary> ::=
     <polygonzm binary representation>

<wkbpolygonpatchz binary> ::=
     <polygonz binary representation>

<wkbpolygonpatchm binary> ::=
     <polygonm binary representation>

<wkbpolygonpatch binary> ::=
     <polygon binary representation>

<wkbtrianglepatchzm binary> ::=
     <trianglezm binary representation>

<wkbtrianglepatchz binary> ::=
     <trianglez binary representation>

<wkbtrianglepatchm binary> ::=
     <trianglem binary representation>

<wkbtrianglepatch binary> ::=
     <triangle binary representation>

<wkbtinelement binary> ::=
     <tinelement binary representation>

<wkbcurvezm binary> ::=
     <linestringzm binary representation>
   | <circularstringzm binary representation>
   | <circlezm binary representation>
   | <geodesiczm binary representation>
   | <ellipticalzm binary representation>
   | <nurbszm binary representation>
   | <clothoidzm binary representation>
   | <spiralzm binary representation>
   | <compoundcurvezm binary representation>

<wkbcurvez binary> ::=
     <linestringz binary representation>
   | <circularstringz binary representation>
   | <circlez binary representation>
   | <geodesicz binary representation>
```

```
   | <ellipticalz binary representation>
   | <nurbsz binary representation>
   | <clothoidz binary representation>
   | <spiralz binary representation>
   | <compoundcurvez binary representation>

<wkbcurvem binary> ::=
     <linestringm binary representation>
   | <circularstringm binary representation>
   | <circlem binary representation>
   | <geodesicm binary representation>
   | <ellipticalm binary representation>
   | <nurbsm binary representation>
   | <clothoidm binary representation>
   | <spiralm binary representation>
   | <compoundcurvem binary representation>

<wkbcurve binary> ::=
     <linestring binary representation>
   | <circularstring binary representation>
   | <circle binary representation>
   | <geodesic binary representation>
   | <elliptical binary representation>
   | <nurbs binary representation>
   | <clothoid binary representation>
   | <spiral binary representation>
   | <compoundcurve binary representation>

<wkbringzm binary> ::=
     <linestringzm binary representation>
   | <circularstringzm binary representation>
   | <circlezm binary representation>
   | <geodesiczm binary representation>
   | <ellipticalzm binary representation>
   | <nurbszm binary representation>
   | <clothoidzm binary representation>
   | <spiralzm binary representation>
   | <compoundcurvezm binary representation>

<wkbringz binary> ::=
     <linestringz binary representation>
   | <circularstringz binary representation>
   | <circlez binary representation>
   | <geodesicz binary representation>
   | <ellipticalz binary representation>
   | <nurbsz binary representation>
   | <clothoidz binary representation>
   | <spiralz binary representation>
   | <compoundcurvez binary representation>

<wkbringm binary> ::=
     <linestringm binary representation>
   | <circularstringm binary representation>
   | <circlem binary representation>
   | <geodesicm binary representation>
   | <ellipticalm binary representation>
   | <nurbsm binary representation>
   | <clothoidm binary representation>
   | <spiralm binary representation>
   | <compoundcurvem binary representation>

<wkbring binary> ::=
     <linestring binary representation>
   | <circularstring binary representation>
```

```
    | <circle binary representation>
    | <geodesic binary representation>
    | <elliptical binary representation>
    | <nurbs binary representation>
    | <clothoid binary representation>
    | <spiral binary representation>
    | <compoundcurve binary representation>

<wkbshellz binary> ::=
      <polyhedralsurfacez binary representation>
    | <polyhedralsurfacezm binary representation>
    | <compoundsurfacez binary representation>
    | <compoundsurfacezm binary representation>

<wkbpointzm binary> ::= <wkbx> <wkby> <wkbz> <wkbm>

<wkbpointz binary> ::= <wkbx> <wkby> <wkbz>

<wkbpointm binary> ::= <wkbx> <wkby> <wkbm>

<wkbpoint binary> ::= <wkbx> <wkby>

<wkbx> ::= <double>

<wkby> ::= <double>

<wkbz> ::= <double>

<wkbm> ::= <double>

<num> ::= <uint32>

<nume> ::= <uint32>

<wkbmaxsidelength> ::= <double>

<wkbuaxislength> ::= <double>

<wkbvaxislength> ::= <double>

<wkbstartangle> ::= <double>

<wkbendangle> ::= <double>

<wkbstartm> ::= <double>

<wkbendm> ::= <double>

<wkbdegree> ::= <byte>

<wkbweight> ::= <double>

<wkbvalue> ::= <double>

<wkbmultiplicity> ::= <byte>

<wkbscalefactor> ::= <double>

<wkbstartdistance> ::= <double>

<wkbenddistance> ::= <double>

<wkbspirallength> ::= <double>

<wkbstartcurvature> ::= <double>

<wkbendcurvature> ::= <double>

<wkbspiraltype> ::= <byte> <letters>

<wkblinearringzm> ::= <num> <wkbpointzm binary>...

<wkblinearringz> ::= <num> <wkbpointz binary>...

<wkblinearringm> ::= <num> <wkbpointm binary>...

<wkblinearring> ::= <num> <wkbpoint binary>...
```



```
<wkbpointzm> ::= <uint32>
<wkbpointz> ::= <uint32>
<wkbpointm> ::= <uint32>
<wkbpoint> ::= <uint32>
<wkblinestringzm> ::= <uint32>
<wkblinestringz> ::= <uint32>
<wkblinestringm> ::= <uint32>
<wkblinestring> ::= <uint32>
<wkbcircularstringzm> ::= <uint32>
<wkbcircularstringz> ::= <uint32>
<wkbcircularstringm> ::= <uint32>
<wkbcircularstring> ::= <uint32>
<wkbcirclezm> ::= <uint32>
<wkbcirclez> ::= <uint32>
<wkbcirclem> ::= <uint32>
<wkbcircle> ::= <uint32>
<wkbgeodesiczm> ::= <uint32>
<wkbgeodesicz> ::= <uint32>
<wkbgeodesicm> ::= <uint32>
<wkbgeodesic> ::= <uint32>
<wkbellipticalzm> ::= <uint32>
<wkbellipticalz> ::= <uint32>
<wkbellipticalm> ::= <uint32>
<wkbelliptical> ::= <uint32>
<wkbnurbszm> ::= <uint32>
<wkbnurbsz> ::= <uint32>
<wkbnurbsm> ::= <uint32>
<wkbnurbs> ::= <uint32>
<wkbclothoidzm> ::= <uint32>
<wkbclothoidz> ::= <uint32>
<wkbclothoidm> ::= <uint32>
<wkbclothoid> ::= <uint32>
<wkbspiralzm> ::= <uint32>
<wkbspiralz> ::= <uint32>
<wkbspiralm> ::= <uint32>
<wkbspiral> ::= <uint32>
<wkbcompoundcurvezm> ::= <uint32>
<wkbcompoundcurvez> ::= <uint32>
<wkbcompoundcurvem> ::= <uint32>
<wkbcompoundcurve> ::= <uint32>
```

```
<wkbtrianglezm> ::= <uint32>
<wkbtrianglez> ::= <uint32>
<wkbtrianglem> ::= <uint32>
<wkbtriangle> ::= <uint32>
<wkbpolygonzm> ::= <uint32>
<wkbpolygonz> ::= <uint32>
<wkbpolygonm> ::= <uint32>
<wkbpolygon> ::= <uint32>
<wkbcurvepolygonzm> ::= <uint32>
<wkbcurvepolygonz> ::= <uint32>
<wkbcurvepolygonm> ::= <uint32>
<wkbcurvepolygon> ::= <uint32>
<wkbtinzm> ::= <uint32>
<wkbtinz> ::= <uint32>
<wkbtinm> ::= <uint32>
<wkbtin> ::= <uint32>
<wkbpolyhedralsurfacezm> ::= <uint32>
<wkbpolyhedralsurfacez> ::= <uint32>
<wkbpolyhedralsurfacem> ::= <uint32>
<wkbpolyhedralsurface> ::= <uint32>
<wkbcompoundsurfacezm> ::= <uint32>
<wkbcompoundsurfacez> ::= <uint32>
<wkbcompoundsurfacem> ::= <uint32>
<wkbcompoundsurface> ::= <uint32>
<wkbbrepsolidz> ::= <uint32>
<wkbmultipointzm> ::= <uint32>
<wkbmultipointz> ::= <uint32>
<wkbmultipointm> ::= <uint32>
<wkbmultipoint> ::= <uint32>
<wkbmultilinestringzm> ::= <uint32>
<wkbmultilinestringz> ::= <uint32>
<wkbmultilinestringm> ::= <uint32>
<wkbmultilinestring> ::= <uint32>
<wkbmulticurvezm> ::= <uint32>
<wkbmulticurvez> ::= <uint32>
<wkbmulticurvem> ::= <uint32>
<wkbmulticurve> ::= <uint32>
<wkbmultisurfacezm> ::= <uint32>
<wkbmultisurfacez> ::= <uint32>
<wkbmultisurfacem> ::= <uint32>
```

```
<wkbmultisurface> ::= <uint32>

<wkbmultipolygonzm> ::= <uint32>

<wkbmultipolygonz> ::= <uint32>

<wkbmultipolygonm> ::= <uint32>

<wkbmultipolygon> ::= <uint32>

<wkbgeometrycollectionzm> ::= <uint32>

<wkbgeometrycollectionz> ::= <uint32>

<wkbgeometrycollectionm> ::= <uint32>

<wkbgeometrycollection> ::= <uint32>

<wkbaffineplacementz> ::= <uint32>

<wkbaffineplacement> ::= <uint32>

<byte order> ::=
     <big endian>
   | <little endian>

<big endian> ::= !! See Description

<little endian> ::= !! See Description

<byte> ::= !! See Description

<uint32> ::= !! See Description

<double> ::= !! See Description

<bit> ::= !! See Description
```

a) Case:

   i) If <well-known binary representation> immediately contains a <well-knownzm binary representation>, then <well-known binary representation> produces an ST_Geometry value specified by the immediately contained <well-knownzm binary representation>.

   ii) If <well-known binary representation> immediately contains a <well-knownz binary representation>, then <well-known binary representation> produces an ST_Geometry value specified by the immediately contained <well-knownz binary representation>.

   iii) If <well-known binary representation> immediately contains a <well-knownm binary representation>, then <well-known binary representation> produces an ST_Geometry value specified by the immediately contained <well-knownm binary representation>.

   iv) Otherwise, <well-known binary representation> produces an ST_Geometry value specified by the immediately contained <well-known2d binary representation>.

b) Case:

   i) If <well-knownzm binary representation> immediately contains a <pointzm binary representation>, then <well-knownzm binary representation> produces an *ST_Point* value specified by the immediately contained <pointzm binary representation>.

   ii) If <well-knownzm binary representation> immediately contains a <curvezm binary representation>, then <well-knownzm binary representation> produces an *ST_Curve* value specified by the immediately contained <curvezm binary representation>.

   iii) If <well-knownzm binary representation> immediately contains a <surfacezm binary representation>, then <well-knownzm binary representation> produces an *ST_Surface* value specified by the immediately contained <surfacezm binary representation>.

   iv) Otherwise, <well-knownzm binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <collectionzm binary representation>.

c) Case:

i) If <well-knownz binary representation> immediately contains a <pointz binary representation>, then <well-knownz binary representation> produces an *ST_Point* value specified by the immediately contained <pointz binary representation>.

ii) If <well-knownz binary representation> immediately contains a <curvez binary representation>, then <well-knownz binary representation> produces an *ST_Curve* value specified by the immediately contained <curvez binary representation>.

iii) If <well-knownz binary representation> immediately contains a <surfacez binary representation>, then <well-knownz binary representation> produces an *ST_Surface* value specified by the immediately contained <surfacez binary representation>.

iv) If <well-knownz binary representation> immediately contains a <solidz binary representation>, then <well-knownz binary representation> produces an *ST_Solid* value specified by the immediately contained <solidz binary representation>.

v) Otherwise, <well-knownz binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <collectionz binary representation>.

d) Case:

i) If <well-knownm binary representation> immediately contains a <pointm binary representation>, then <well-knownm binary representation> produces an *ST_Point* value specified by the immediately contained <pointm binary representation>.

ii) If <well-knownm binary representation> immediately contains a <curvem binary representation>, then <well-knownm binary representation> produces an *ST_Curve* value specified by the immediately contained <curvem binary representation>.

iii) If <well-knownm binary representation> immediately contains a <surfacem binary representation>, then <well-knownm binary representation> produces an *ST_Surface* value specified by the immediately contained <surfacem binary representation>.

iv) Otherwise, <well-knownm binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <collectionm binary representation>.

e) Case:

i) If <well-known2d binary representation> immediately contains a <point binary representation>, then <well-known2d binary representation> produces an *ST_Point* value specified by the immediately contained <point binary representation>.

ii) If <well-known2d binary representation> immediately contains a <curve binary representation>, then <well-known2d binary representation> produces an *ST_Curve* value specified by the immediately contained <curve binary representation>.

iii) If <well-known2d binary representation> immediately contains a <surface binary representation>, then <well-known2d binary representation> produces an *ST_Surface* value specified by the immediately contained <surface binary representation>.

iv) Otherwise, <well-known2d binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <collection binary representation>.

f) Case:

i) If <pointzm binary representation> immediately contains a <wkbpointzm binary>, then <pointzm binary representation> is the well-known binary representation for an *ST_Point* value that is produced by <wkbpointzm binary>.

ii) Otherwise, <pointzm binary representation> produces an empty set of type *ST_Point*.

g) Case:

i) If <pointz binary representation> immediately contains a <wkbpointz binary>, then <pointz binary representation> is the well-known binary representation for an *ST_Point* value that is produced by <wkbpointz binary>.

ii) Otherwise, <pointz binary representation> produces an empty set of type *ST_Point*.

h) Case:

i) If <pointm binary representation> immediately contains a <wkbpointm binary>, then <pointm binary representation> is the well-known binary representation for an *ST_Point* value that is produced by <wkbpointm binary>.

ii) Otherwise, <pointm binary representation> produces an empty set of type *ST_Point.*

i) Case:

i) If <point binary representation> immediately contains a <wkbpoint binary>, then <point binary representation> is the well-known binary representation for an *ST_Point* value that is produced by <wkbpoint binary>.

ii) Otherwise, <point binary representation> produces an empty set of type *ST_Point.*

j) Case:

i) If <curvezm binary representation> immediately contains a <linestringzm binary representation>, then <curvezm binary representation> produces an *ST_LineString* value specified by the immediately contained <linestringzm binary representation>.

ii) If <curvezm binary representation> immediately contains a <circularstringzm binary representation>, then <curvezm binary representation> produces an *ST_CircularString* value specified by the immediately contained <circularstringzm binary representation>.

iii) If <curvezm binary representation> immediately contains a <circlezm binary representation>, then <curvezm binary representation> produces an *ST_Circle* value specified by the immediately contained <circlezm binary representation>.

iv) If <curvezm binary representation> immediately contains a <geodesiczm binary representation>, then <curvezm binary representation> produces an *ST_GeodesicString* value specified by the immediately contained <geodesiczm binary representation>.

v) If <curvezm binary representation> immediately contains a <ellipticalzm binary representation>, then <curvezm binary representation> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalzm binary representation>.

vi) If <curvezm binary representation> immediately contains a <nurbszm binary representation>, then <curvezm binary representation> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbszm binary representation>.

vii) If <curvezm binary representation> immediately contains a <clothoidzm binary representation>, then <curvezm binary representation> produces an *ST_Clothoid* value specified by the immediately contained <clothoidzm binary representation>.

viii) If <curvezm binary representation> immediately contains a <spiralzm binary representation>, then <curvezm binary representation> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralzm binary representation>.

ix) Otherwise, <curvezm binary representation> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvezm binary representation>.

k) Case:

i) If <curvez binary representation> immediately contains a <linestringz binary representation>, then <curvez binary representation> produces an *ST_LineString* value specified by the immediately contained <linestringz binary representation>.

ii) If <curvez binary representation> immediately contains a <circularstringz binary representation>, then <curvez binary representation> produces an *ST_CircularString* value specified by the immediately contained <circularstringz binary representation>.

iii) If <curvez binary representation> immediately contains a <circlez binary representation>, then <curvez binary representation> produces an *ST_Circle* value specified by the immediately contained <circlez binary representation>.

iv) If <curvez binary representation> immediately contains a <geodesicz binary representation>, then <curvez binary representation> produces an *ST_GeodesicString* value specified by the immediately contained <geodesicz binary representation>.

v) If <curvez binary representation> immediately contains a <ellipticalz binary representation>, then <curvez binary representation> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalz binary representation>.

vi) If <curvez binary representation> immediately contains a <nurbsz binary representation>, then <curvez binary representation> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbsz binary representation>.

vii) If <curvez binary representation> immediately contains a <clothoidz binary representation>, then <curvez binary representation> produces an *ST_Clothoid* value specified by the immediately contained <clothoidz binary representation>.

viii) If <curvez binary representation> immediately contains a <spiralz binary representation>, then <curvez binary representation> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralz binary representation>.

ix) Otherwise, <curvez binary representation> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvez binary representation>.

l) Case:

i) If <curvem binary representation> immediately contains a <linestringm binary representation>, then <curvem binary representation> produces an *ST_LineString* value specified by the immediately contained <linestringm binary representation>.

ii) If <curvem binary representation> immediately contains a <circularstringm binary representation>, then <curvem binary representation> produces an *ST_CircularString* value specified by the immediately contained <circularstringm binary representation>.

iii) If <curvem binary representation> immediately contains a <circlem binary representation>, then <curvem binary representation> produces an *ST_Circle* value specified by the immediately contained <circlem binary representation>.

iv) If <curvem binary representation> immediately contains a <geodesicm binary representation>, then <curvem binary representation> produces an *ST_GeodesicString* value specified by the immediately contained <geodesicm binary representation>.

v) If <curvem binary representation> immediately contains a <ellipticalm binary representation>, then <curvem binary representation> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalm binary representation>.

vi) If <curvem binary representation> immediately contains a <nurbsm binary representation>, then <curvem binary representation> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbsm binary representation>.

vii) If <curvem binary representation> immediately contains a <clothoidm binary representation>, then <curvem binary representation> produces an *ST_Clothoid* value specified by the immediately contained <clothoidm binary representation>.

viii) If <curvem binary representation> immediately contains a <spiralm binary representation>, then <curvem binary representation> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralm binary representation>.

ix) Otherwise, <curvem binary representation> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvem binary representation>.

m) Case:

i) If <curve binary representation> immediately contains a <linestring binary representation>, then <curve binary representation> produces an *ST_LineString* value specified by the immediately contained <linestring binary representation>.

ii) If <curve binary representation> immediately contains a <circularstring binary representation>, then <curve binary representation> produces an *ST_CircularString* value specified by the immediately contained <circularstring binary representation>.

iii) If <curve binary representation> immediately contains a <circle binary representation>, then <curve binary representation> produces an *ST_Circle* value specified by the immediately contained <circle binary representation>.

iv) If <curve binary representation> immediately contains a <geodesic binary representation>, then <curve binary representation> produces an *ST_GeodesicString* value specified by the immediately contained <geodesic binary representation>.

v) If <curve binary representation> immediately contains a <elliptical binary representation>, then <curve binary representation> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical binary representation>.

vi) If <curve binary representation> immediately contains a <nurbs binary representation>, then <curve binary representation> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs binary representation>.

vii) If <curve binary representation> immediately contains a <clothoid binary representation>, then <curve binary representation> produces an *ST_Clothoid* value specified by the immediately contained <clothoid binary representation>.

viii) If <curve binary representation> immediately contains a <spiral binary representation>, then <curve binary representation> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral binary representation>.

ix) Otherwise, <curve binary representation> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurve binary representation>.

n) Case:

i) If <linestringzm binary representation> immediately contains <num>, then <linestringzm binary representation> is the well-known binary representation for an *ST_LineString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointzm binary>s. <linestringzm binary representation> produces an *ST_LineString* value as the result of the value expression: *NEW ST_LineString(APA)*.

ii) Otherwise, <linestringzm binary representation> produces an empty set of type *ST_LineString*.

o) Case:

i) If <linestringz binary representation> immediately contains <num>, then <linestringz binary representation> is the well-known binary representation for an *ST_LineString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointz binary>s. <linestringz binary representation> produces an *ST_LineString* value as the result of the value expression: *NEW ST_LineString(APA)*.

ii) Otherwise, <linestringz binary representation> produces an empty set of type *ST_LineString*.

p) Case:

i) If <linestringm binary representation> immediately contains <num>, then <linestringm binary representation> is the well-known binary representation for an *ST_LineString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointm binary>s. <linestringm binary representation> produces an *ST_LineString* value as the result of the value expression: *NEW ST_LineString(APA)*.

ii) Otherwise, <linestringm binary representation> produces an empty set of type *ST_LineString*.

q) Case:

i) If <linestring binary representation> immediately contains <num>, then <linestring binary representation> is the well-known binary representation for an *ST_LineString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpoint binary>s. <linestring binary representation> produces an *ST_LineString* value as the result of the value expression: *NEW ST_LineString(APA)*.

ii) Otherwise, <linestring binary representation> produces an empty set of type *ST_LineString*.

r) Case:

i) If <circularstringzm binary representation> immediately contains <num>, then <circularstringzm binary representation> is the well-known binary representation for an *ST_CircularString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointzm binary>s. <circularstringzm binary representation> produces an *ST_CircularString* value as the result of the value expression: *NEW ST_CircularString(APA)*.

ii) Otherwise, <circularstringzm binary representation> produces an empty set of type *ST_CircularString*.

s) Case:

i) If <circularstringz binary representation> immediately contains <num>, then <circularstringz binary representation> is the well-known binary representation for an *ST_CircularString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointz binary>s. <circularstringz binary representation> produces an *ST_CircularString* value as the result of the value expression: *NEW ST_CircularString(APA)*.

ii) Otherwise, <circularstringz binary representation> produces an empty set of type *ST_CircularString*.

t) Case:

i) If <circularstringm binary representation> immediately contains <num>, then <circularstringm binary representation> is the well-known binary representation for an *ST_CircularString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointm binary>s. <circularstringm binary representation> produces an *ST_CircularString* value as the result of the value expression: *NEW ST_CircularString(APA)*.

ii) Otherwise, <circularstringm binary representation> produces an empty set of type *ST_CircularString*.

u) Case:

i) If <circularstring binary representation> immediately contains <num>, then <circularstring binary representation> is the well-known binary representation for an *ST_CircularString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpoint binary>s. <circularstring binary representation> produces an *ST_CircularString* value as the result of the value expression: *NEW ST_CircularString(APA)*.

ii) Otherwise, <circularstring binary representation> produces an empty set of type *ST_CircularString*.

v) Case:

i) If <circlezm binary representation> immediately contains <num>, then <circlezm binary representation> is the well-known binary representation for an *ST_Circle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointzm binary>s. <circlezm binary representation> produces an *ST_Circle* value as the result of the value expression: *NEW ST_Circle(APA)*.

ii) Otherwise, <circlezm binary representation> produces an empty set of type *ST_Circle*.

w) Case:

i) If <circlez binary representation> immediately contains <num>, then <circlez binary representation> is the well-known binary representation for an *ST_Circle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointz binary>s. <circlez binary representation> produces an *ST_Circle* value as the result of the value expression: *NEW ST_Circle(APA)*.

ii) Otherwise, <circlez binary representation> produces an empty set of type *ST_Circle*.

x) Case:

i) If <circlem binary representation> immediately contains <num>, then <circlem binary representation> is the well-known binary representation for an *ST_Circle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointm binary>s. <circlem binary representation> produces an *ST_Circle* value as the result of the value expression: *NEW ST_Circle(APA)*.

ii) Otherwise, <circlem binary representation> produces an empty set of type *ST_Circle*.

y) Case:

i) If <circle binary representation> immediately contains <num>, then <circle binary representation> is the well-known binary representation for an *ST_Circle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpoint binary>s. <circle binary representation> produces an *ST_Circle* value as the result of the value expression: *NEW ST_Circle(APA)*.

ii) Otherwise, <circle binary representation> produces an empty set of type *ST_Circle*.

z) Case:

i) If <geodesiczm binary representation> immediately contains <num>, then <geodesiczm binary representation> is the well-known binary representation for an *ST_GeodesicString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointzm binary>s. <geodesiczm binary representation> produces an *ST_GeodesicString* value as the result of the value expression: *NEW ST_GeodesicString(APA)*.

ii) Otherwise, <geodesiczm binary representation> produces an empty set of type *ST_GeodesicString*.

aa) Case:

i) If <geodesicz binary representation> immediately contains <num>, then <geodesicz binary representation> is the well-known binary representation for an *ST_GeodesicString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointz binary>s. <geodesicz binary representation> produces an *ST_GeodesicString* value as the result of the value expression: *NEW ST_GeodesicString(APA)*.

ii) Otherwise, <geodesicz binary representation> produces an empty set of type *ST_GeodesicString*.

ab) Case:

i) If <geodesicm binary representation> immediately contains <num>, then <geodesicm binary representation> is the well-known binary representation for an *ST_GeodesicString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointm binary>s. <geodesicm binary representation> produces an *ST_GeodesicString* value as the result of the value expression: *NEW ST_GeodesicString(APA)*.

ii) Otherwise, <geodesicm binary representation> produces an empty set of type *ST_GeodesicString*.

ac) Case:

i) If <geodesic binary representation> immediately contains <num>, then <geodesic binary representation> is the well-known binary representation for an *ST_GeodesicString* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpoint binary>s. <geodesic binary representation> produces an *ST_GeodesicString* value as the result of the value expression: *NEW ST_GeodesicString(APA)*.

ii) Otherwise, <geodesic binary representation> produces an empty set of type *ST_GeodesicString*.

ad) Case:

i) If <ellipticalzm binary representation> immediately contains <wkbreferencelocationzm binary>, then <ellipticalzm binary representation> is the well-known binary representation for an *ST_EllipticalCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationzm binary>. Let *UAL* be a DOUBLE PRECISION u axis length specified by the immediately contained <wkbuaxislength>. Let *VAL* be a DOUBLE PRECISION v axis length specified by the immediately contained <wkbvaxislength>. Let *SA* be an *ST_Angle* value start angle specified by the immediately contained <wkbstartangle>. Let *EA* be an *ST_Angle* value end angle specified by the immediately contained <wkbendangle>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <ellipticalzm binary representation> produces an *ST_EllipticalCurve* value as the result of the value expression: *NEW ST_EllipticalCurve(RL, UAL, VAL, SA, EA, SM, EM)*.

ii) Otherwise, <ellipticalzm binary representation> produces an empty set of type *ST_EllipticalCurve*.

ae) Case:

i) If <ellipticalz binary representation> immediately contains <wkbreferencelocationz binary>, then <ellipticalz binary representation> is the well-known binary representation for an *ST_EllipticalCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationz binary>. Let *UAL* be a DOUBLE PRECISION u axis length specified by the immediately contained <wkbuaxislength>. Let *VAL* be a DOUBLE PRECISION v axis length specified by the immediately contained <wkbvaxislength>. Let *SA* be an *ST_Angle* value start angle specified by the immediately contained <wkbstartangle>. Let *EA* be an *ST_Angle* value end angle specified by the immediately contained <wkbendangle>. <ellipticalz binary representation> produces an *ST_EllipticalCurve* value as the result of the value expression: *NEW ST_EllipticalCurve(RL, UAL, VAL, SA, EA)*.

ii) Otherwise, <ellipticalz binary representation> produces an empty set of type *ST_EllipticalCurve*.

af) Case:

i) If <ellipticalm binary representation> immediately contains <wkbreferencelocationm binary>, then <ellipticalm binary representation> is the well-known binary representation for an *ST_EllipticalCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationm binary>. Let *UAL* be a DOUBLE PRECISION u axis length specified by the immediately contained <wkbuaxislength>. Let *VAL* be a DOUBLE PRECISION v axis length specified by the immediately contained <wkbvaxislength>. Let *SA* be an *ST_Angle* value start angle specified by the immediately contained <wkbstartangle>. Let *EA* be an *ST_Angle* value end angle specified by the immediately contained <wkbendangle>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <ellipticalm binary representation> produces an *ST_EllipticalCurve* value as the result of the value expression: *NEW ST_EllipticalCurve(RL, UAL, VAL, SA, EA, SM, EM)*.

ii) Otherwise, <ellipticalm binary representation> produces an empty set of type *ST_EllipticalCurve*.

ag) Case:

i) If <elliptical binary representation> immediately contains <wkbreferencelocation binary>, then <elliptical binary representation> is the well-known binary representation for an *ST_EllipticalCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocation binary>. Let *UAL* be a DOUBLE PRECISION u axis length specified by the immediately contained <wkbuaxislength>. Let *VAL* be a DOUBLE PRECISION v axis length specified by the immediately contained <wkbvaxislength>. Let *SA* be an *ST_Angle* value start angle specified by the immediately contained <wkbstartangle>. Let *EA* be an *ST_Angle* value end angle specified by the immediately contained <wkbendangle>. <elliptical binary representation> produces an *ST_EllipticalCurve* value as the result of the value expression: *NEW ST_EllipticalCurve(RL, UAL, VAL, SA, EA)*.

ii) Otherwise, <elliptical binary representation> produces an empty set of type *ST_EllipticalCurve*.

ah) <wkbreferencelocationzm binary> produces an *ST_AffinePlacement* value specified by the immediately contained <affineplacementz binary representation>.

ai) <wkbreferencelocationz binary> produces an *ST_AffinePlacement* value specified by the immediately contained <affineplacementz binary representation>.

aj) <wkbreferencelocationm binary> produces an *ST_AffinePlacement* value specified by the immediately contained <affineplacement binary representation>.

ak) <wkbreferencelocation binary> produces an *ST_AffinePlacement* value specified by the immediately contained <affineplacement binary representation>.

al) Case:

i) If <affineplacementz binary representation> immediately contains <wkblocationz>, then <affineplacementz binary representation> is the well-known binary representation for an *ST_AffinePlacement* value. Let *L* be an *ST_Point* value location specified by the immediately contained <wkblocationz>. Let *RD* be an *ST_Vector* ARRAY value with a cardinality of <numd> specified by the immediately contained <wkbreferencedirectionsz>. <affineplacementz binary representation> produces an *ST_AffinePlacement* value as the result of the value expression: *NEW ST_AffinePlacement(L, RD)*.

ii) Otherwise, <affineplacementz binary representation> produces an empty set of type *ST_AffinePlacement*.

am) Case:

i) If <affineplacement binary representation> immediately contains <wkblocation>, then <affineplacement binary representation> is the well-known binary representation for an *ST_AffinePlacement* value. Let *L* be an *ST_Point* value location specified by the immediately contained <wkblocation>. Let *RD* be an *ST_Vector* ARRAY value with a cardinality of <numd> specified by the immediately contained <wkbreferencedirections>. <affineplacement binary representation> produces an *ST_AffinePlacement* value as the result of the value expression: *NEW ST_AffinePlacement(L, RD)*.

ii) Otherwise, <affineplacement binary representation> produces an empty set of type *ST_AffinePlacement*.

an) <wkblocationz> specifies an *ST_Point* value producible by <wkbpointz binary>.

ao) <wkblocation> specifies an *ST_Point* value producible by <wkbpoint binary>.

ap) <wkbreferencedirectionsz> produces an *ST_Vector* ARRAY value containing <num> *ST_Vector* values producible by the immediately contained <wkbvectorz binary>s.

aq) <wkbreferencedirections> produces an *ST_Vector* ARRAY value containing <num> *ST_Vector* values producible by the immediately contained <wkbvector binary>s.

ar) Case:

i) If <nurbszm binary representation> immediately contains <wkbdegree>, then <nurbszm binary representation> is the well-known binary representation for an *ST_NURBSCurve* value. Let *D* be an INTEGER value degree specified by the immediately contained <wkbdegree>. Let *CP* be an *ST_NURBSPoint* ARRAY control points attribute value specified by the immediately contained <wkbcontrolpointsz binary>. Let *K* be an *ST_Knots* ARRAY knots attribute value specified by the immediately contained <wkbknots binary>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <nurbszm binary representation> produces an *ST_NURBSCurve* value as the result of the value expression: *NEW ST_NURBSCurve(D, CP, K, SM, EM)*.

ii) Otherwise, <nurbszm binary representation> produces an empty set of type *ST_NURBSCurve*.

as) Case:

i) If <nurbsz binary representation> immediately contains <wkbdegree>, then <nurbsz binary representation> is the well-known binary representation for an *ST_NURBSCurve* value. Let *D* be an INTEGER value degree specified by the immediately contained <wkbdegree>. Let *CP* be an *ST_NURBSPoint* ARRAY control points attribute value specified by the immediately contained <wkbcontrolpointsz binary>. Let *K* be an *ST_Knots* ARRAY knots attribute value specified by the immediately contained <wkbknots binary>. <nurbsz binary representation> produces an *ST_NURBSCurve* value as the result of the value expression: *NEW ST_NURBSCurve(D, CP, K)*.

ii) Otherwise, <nurbsz binary representation> produces an empty set of type *ST_NURBSCurve*.

at) Case:

i) If <nurbsm binary representation> immediately contains <wkbdegree>, then <nurbsm binary representation> is the well-known binary representation for an *ST_NURBSCurve* value. Let *D* be an INTEGER value degree specified by the immediately contained <wkbdegree>. Let *CP* be an *ST_NURBSPoint* ARRAY control points attribute value specified by the immediately contained <wkbcontrolpoints binary>. Let *K* be an *ST_Knots* ARRAY knots attribute value specified by the immediately contained <wkbknots binary>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <nurbsm binary representation> produces an *ST_NURBSCurve* value as the result of the value expression: *NEW ST_NURBSCurve(D, CP, K, SM, EM)*.

ii) Otherwise, <nurbsm binary representation> produces an empty set of type *ST_NURBSCurve*.

au) Case:

i) If <nurbs binary representation> immediately contains <wkbdegree>, then <nurbs binary representation> is the well-known binary representation for an *ST_NURBSCurve* value. Let *D* be an INTEGER value degree specified by the immediately contained <wkbdegree>. Let *CP* be an *ST_NURBSPoint* ARRAY control points attribute value specified by the immediately contained <wkbcontrolpoints binary>. Let *K* be an *ST_Knots* ARRAY knots attribute value specified by the immediately contained <wkbknots binary>. <nurbs binary representation> produces an *ST_NURBSCurve* value as the result of the value expression: *NEW ST_NURBSCurve(D, CP, K)*.

ii) Otherwise, <nurbs binary representation> produces an empty set of type *ST_NURBSCurve*.

av) <wkbcontrolpointsz binary> produces an *ST_NURBSPoint* ARRAY value containing <num> *ST_NURBSPoint* values producible by the immediately contained <nurbspointz binary representation>s.

aw) <wkbcontrolpoints binary> produces an *ST_NURBSPoint* ARRAY value containing <num> *ST_NURBSPoint* values producible by the immediately contained <nurbspoint binary representation>s.

ax) Case:

i) If <nurbspointz binary representation> immediately contains <wkbweightedpointz>, then <nurbspointz binary representation> is the well-known binary representation for an *ST_NURBSPoint* value. Let *WP* be an *ST_Point* weighted point attribute value specified by the immediately contained <wkbweightedpointz>. Let *W* be a NULL DOUBLE PRECISION weight attribute value if <bit> = 0 (zero). Let *W* be a DOUBLE PRECISION weight attribute value specified by the immediately contained <wkbweight> if <bit> = 1 (one). <nurbspointz binary representation> produces an *ST_NURBSPoint* value as the result of the value expression: *NEW ST_NURBSPoint(WP, W)*.

ii) Otherwise, <nurbspointz binary representation> produces an empty set of type *ST_NURBSPoint*.

ay) Case:

i) If <nurbspoint binary representation> immediately contains <wkbweightedpoint>, then <nurbspoint binary representation> is the well-known binary representation for an *ST_NURBSPoint* value. Let *WP* be an *ST_Point* weighted point attribute value specified by the immediately contained <wkbweightedpoint>. Let *W* be a NULL DOUBLE PRECISION weight attribute value if <bit> = 0 (zero). Let *W* be a DOUBLE PRECISION weight attribute value specified by the immediately contained <wkbweight> if <bit> = 1 (one). <nurbspoint binary representation> produces an *ST_NURBSPoint* value as the result of the value expression: *NEW ST_NURBSPoint(WP, W)*.

ii) Otherwise, <nurbspoint binary representation> produces an empty set of type *ST_NURBSPoint*.

az) <wkbweightedpointz> specifies an *ST_Point* value producible by <wkbpointz binary>.

ba) <wkbweightedpoint> specifies an *ST_Point* value producible by <wkbpoint binary>.

bb) <wkbknots binary> produces an *ST_Knot* ARRAY value containing <num> *ST_Knot* values producible by the immediately contained <knot binary representation>s.

bc) Case:

i) If <knot binary representation> immediately contains <wkbvalue>, then <knot binary representation> is the well-known binary representation for an *ST_Knot* value. Let *V* be a DOUBLE PRECISION value attribute value specified by the immediately contained <wkbvalue>. Let *M* be an INTEGER multiplicity attribute value specified by the immediately contained <wkbmultiplicity>. <knot binary representation> produces an *ST_Knot* value as the result of the value expression: *NEW ST_Knot(V, M)*.

ii) Otherwise, <knot binary representation> produces an empty set of type *ST_Knot*.

bd) Case:

i) If <clothoidzm binary representation> immediately contains <wkbreferencelocationzm binary>, then <clothoidzm binary representation> is the well-known binary representation for an *ST_Clothoid* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationzm binary>. Let *SF* be a DOUBLE PRECISION scale factor specified by the immediately contained <wkbscalefactor>. Let *SD* be a DOUBLE PRECISION start distance specified by the immediately contained <wkbstartdistance>. Let *ED* be a DOUBLE PRECISION value end distance specified by the immediately contained <wkbenddistance>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <clothoidzm binary representation> produces an *ST_Clothoid* value as the result of the value expression: *NEW ST_Clothoid(RL, SF, SD, ED, SM, EM)*.

ii) Otherwise, <clothoidzm binary representation> produces an empty set of type *ST_Clothoid*.

be) Case:

i) If <clothoidz binary representation> immediately contains <wkbreferencelocationz binary>, then <clothoidz binary representation> is the well-known binary representation for an *ST_Clothoid* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationz binary>. Let *SF* be a DOUBLE PRECISION scale factor specified by the immediately contained <wkbscalefactor>. Let *SD* be a DOUBLE PRECISION start distance specified by the immediately contained <wkbstartdistance>. Let *ED* be a DOUBLE PRECISION value end distance specified by the immediately contained <wkbenddistance>. <clothoidz binary representation> produces an *ST_Clothoid* value as the result of the value expression: *NEW ST_Clothoid (RL, SF, SD, ED)*.

ii) Otherwise, <clothoidz binary representation> produces an empty set of type *ST_Clothoid*.

bf) Case:

i) If <clothoidm binary representation> immediately contains <wkbreferencelocationm binary>, then <clothoidm binary representation> is the well-known binary representation for an *ST_Clothoid* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationm binary>. Let *SF* be a DOUBLE PRECISION scale factor specified by the immediately contained <wkbscalefactor>. Let *SD* be a DOUBLE PRECISION start distance specified by the immediately contained <wkbstartdistance>. Let *ED* be a DOUBLE PRECISION value end distance specified by the immediately contained <wkbenddistance>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <clothoidm binary representation> produces an *ST_Clothoid* value as the result of the value expression: *NEW ST_Clothoid(RL, SF, SD, ED, SM, EM)*.

ii) Otherwise, <clothoidm binary representation> produces an empty set of type *ST_Clothoid*.

bg) Case:

i) If <clothoid binary representation> immediately contains <wkbreferencelocation binary>, then <clothoid binary representation> is the well-known binary representation for an *ST_Clothoid* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocation binary>. Let *SF* be a DOUBLE PRECISION scale factor specified by the immediately contained <wkbscalefactor>. Let *SD* be a DOUBLE PRECISION start distance specified by the immediately contained <wkbstartdistance>. Let *ED* be a DOUBLE PRECISION value end distance specified by the immediately contained <wkbenddistance>. <clothoid binary representation> produces an *ST_Clothoid* value as the result of the value expression: *NEW ST_Clothoid(RL, SF, SD, ED)*.

ii) Otherwise, <clothoid binary representation> produces an empty set of type *ST_Clothoid*.

bh) Case:

i) If <spiralzm binary representation> immediately contains <wkbreferencelocationzm binary>, then <spiralzm binary representation> is the well-known binary representation for an *ST_SpiralCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationzm binary>. Let *SL* be a DOUBLE PRECISION spiral length specified by the immediately contained <wkbspirallength>. Let *SC* be a DOUBLE PRECISION start curvature specified by the immediately contained <wkbstartcurvature>. Let *EC* be a DOUBLE PRECISION end curvature specified by the immediately contained <wkbendcurvature>. Let *ST* be a DOUBLE PRECISION value spiral type specified by the immediately contained <wkbspiraltype>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <spiralzm binary representation> produces an *ST_SpiralCurve* value as the result of the value expression: *NEW ST_SpiralCurve(RL, SL, SC, EC, ST, SM, EM)*.

ii) Otherwise, <spiralzm binary representation> produces an empty set of type *ST_SpiralCurve*.

bi) Case:

i) If <spiralz binary representation> immediately contains <wkbreferencelocationz binary>, then <spiralz binary representation> is the well-known binary representation for an *ST_SpiralCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationz binary>. Let *SL* be a DOUBLE PRECISION spiral length specified by the immediately contained <wkbspirallength>. Let *SC* be a DOUBLE PRECISION start curvature specified by the immediately contained <wkbstartcurvature>. Let *EC* be a DOUBLE PRECISION end curvature specified by the immediately contained <wkbendcurvature>. Let *ST* be a DOUBLE PRECISION value spiral type specified by the immediately contained <wkbspiraltype>. <spiralz binary representation> produces an *ST_SpiralCurve* value as the result of the value expression: *NEW ST_SpiralCurve(RL, SL, SC, EC, ST)*.

ii) Otherwise, <spiralz binary representation> produces an empty set of type *ST_SpiralCurve*.

bj) Case:

i) If <spiralm binary representation> immediately contains <wkbreferencelocationm binary>, then <spiralm binary representation> is the well-known binary representation for an *ST_SpiralCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocationm binary>. Let *SL* be a DOUBLE PRECISION spiral length specified by the immediately contained <wkbspirallength>. Let *SC* be a DOUBLE PRECISION start curvature specified by the immediately contained <wkbstartcurvature>. Let *EC* be a DOUBLE PRECISION end curvature specified by the immediately contained <wkbendcurvature>. Let *ST* be a DOUBLE PRECISION value spiral type specified by the immediately contained <wkbspiraltype>. Let *SM* be a DOUBLE PRECISION value start measure specified by the immediately contained <wkbstartm>. Let *EM* be a DOUBLE PRECISION value end measure specified by the immediately contained <wkbendm>. <spiralm binary representation> produces an *ST_SpiralCurve* value as the result of the value expression: *NEW ST_SpiralCurve(RL, SL, SC, EC, ST, SM, EM)*.

ii) Otherwise, <spiralm binary representation> produces an empty set of type *ST_SpiralCurve*.

bk) Case:

i) If <spiral binary representation> immediately contains <wkbreferencelocation binary>, then <spiral binary representation> is the well-known binary representation for an *ST_SpiralCurve* value. Let *RL* be an *ST_AffinePlacement* value reference location specified by the immediately contained <wkbreferencelocation binary>. Let *SL* be a DOUBLE PRECISION spiral length specified by the immediately contained <wkbspirallength>. Let *SC* be a DOUBLE PRECISION start curvature specified by the immediately contained <wkbstartcurvature>. Let *EC* be a DOUBLE PRECISION end curvature specified by the immediately contained <wkbendcurvature>. Let *ST* be a DOUBLE PRECISION value spiral type specified by the immediately contained <wkbspiraltype>. <spiral binary representation> produces an *ST_SpiralCurve* value as the result of the value expression: *NEW ST_SpiralCurve(RL, SL, SC, EC, ST)*.

ii) Otherwise, <spiral binary representation> produces an empty set of type *ST_SpiralCurve*.

bl) Case:

i) If <compoundcurvezm binary representation> immediately contains <num>, then <compoundcurvezm binary representation> is the well-known binary representation for an *ST_CompoundCurve* value. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbcurvezm binary>s. <compoundcurvezm binary representation> produces an *ST_CompoundCurve* value as the result of the value expression: *NEW ST_CompoundCurve(ACA)*.

ii) Otherwise, <compoundcurvezm binary representation> produces an empty set of type *ST_CompoundCurve*.

bm) Case:

i) If <compoundcurvez binary representation> immediately contains <num>, then <compoundcurvez binary representation> is the well-known binary representation for an *ST_CompoundCurve* value. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbcurvez binary>s. <compoundcurvez binary representation> produces an *ST_CompoundCurve* value as the result of the value expression: *NEW ST_CompoundCurve(ACA)*.

ii) Otherwise, <compoundcurvez binary representation> produces an empty set of type *ST_CompoundCurve*.

bn) Case:

i) If <compoundcurvem binary representation> immediately contains <num>, then <compoundcurvem binary representation> is the well-known binary representation for an *ST_CompoundCurve* value. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbcurvem binary>s. <compoundcurvem binary representation> produces an *ST_CompoundCurve* value as the result of the value expression: *NEW ST_CompoundCurve(ACA)*.

ii) Otherwise, <compoundcurvem binary representation> produces an empty set of type *ST_CompoundCurve*.

bo) Case:

i) If <compoundcurve binary representation> immediately contains <num>, then <compoundcurve binary representation> is the well-known binary representation for an *ST_CompoundCurve* value. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbcurve binary>s. <compoundcurve binary representation> produces an *ST_CompoundCurve* value as the result of the value expression: *NEW ST_CompoundCurve(ACA)*.

ii) Otherwise, <compoundcurve binary representation> produces an empty set of type *ST_CompoundCurve*.

bp) Case:

i) If <surfacezm binary representation> immediately contains a <curvepolygonzm binary representation>, then <surfacezm binary representation> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygonzm binary representation>.

ii) If <surfacezm binary representation> immediately contains a <polyhedralsurfacezm binary representation>, then <surfacezm binary representation> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurfacezm binary representation>.

iii) Otherwise, <surfacezm binary representation> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurfacezm binary representation>.

bq) Case:

i) If <surfacez binary representation> immediately contains a <curvepolygonz binary representation>, then <surfacez binary representation> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygonz binary representation>.

ii) If <surfacez binary representation> immediately contains a <polyhedralsurfacez binary representation>, then <surfacez binary representation> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurfacez binary representation>.

iii) Otherwise, <surfacez binary representation> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurfacez binary representation>.

br) Case:

i) If <surfacem binary representation> immediately contains a <curvepolygonm binary representation>, then <surfacem binary representation> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygonm binary representation>.

ii) If <surfacem binary representation> immediately contains a <polyhedralsurfacem binary representation>, then <surfacem binary representation> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurfacem binary representation>.

iii) Otherwise, <surfacem binary representation> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurfacem binary representation>.

bs) Case:

i) If <surface binary representation> immediately contains a <curvepolygon binary representation>, then <surface binary representation> produces an *ST_CurvePolygon* value specified by the immediately contained <curvepolygonm binary representation>.

ii) If <surface binary representation> immediately contains a <polyhedralsurface binary representation>, then <surface binary representation> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurface binary representation>.

iii) Otherwise, <surface binary representation> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurface binary representation>.

bt) Case:

i) If <curvepolygonzm binary representation> immediately contains a <num>, then <curvepolygonzm binary representation> produces an *ST_CurvePolygon*. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbringzm binary>s.

Case:

1) If the cardinality of *ACA* is 0 (zero), then <curvepolygonzm binary representation> produces an empty set of type *ST_CurvePolygon*.

2) If the cardinality of *ACA* is 1 (one), then let *AER* be the element of *ACA*. <curvepolygonzm binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER)*.

3) Otherwise, let *AER* be the first element in *ACA* and let *AIR* be the sublist of *ACA* containing the other elements of *ACA*. <curvepolygonzm binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER, AIR)*.

ii) If <curvepolygonzm binary representation> immediately contains a <polygonzm binary representation>, then <curvepolygonzm binary representation> produces an *ST_Polygon* value specified by the immediately contained <polygonzm binary representation>.

iii) Otherwise, <curvepolygonzm binary representation> produces an empty set of type *ST_CurvePolygon*.

bu) Case:

i) If <curvepolygonz binary representation> immediately contains a <num>, then <curvepolygonz binary representation> produces an *ST_CurvePolygon*. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbringz binary>s.

Case:

1) If the cardinality of *ACA* is 0 (zero), then <curvepolygonz binary representation> produces an empty set of type *ST_CurvePolygon*.

2) If the cardinality of *ACA* is 1 (one), then let *AER* be the element of *ACA*. <curvepolygonz binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER)*.

3) Otherwise, let *AER* be the first element in *ACA* and let *AIR* be the sublist of *ACA* containing the other elements of *ACA*. <curvepolygonz binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER, AIR)*.

ii) If <curvepolygonz binary representation> immediately contains a <polygonz binary representation>, then <curvepolygonz binary representation> produces an *ST_Polygon* value specified by the immediately contained <polygonz binary representation>.

iii) Otherwise, <curvepolygonz binary representation> produces an empty set of type *ST_CurvePolygon*.

bv) Case:

i) If <curvepolygonm binary representation> immediately contains a <num>, then <curvepolygonm binary representation> produces an *ST_CurvePolygon*. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbringm binary>s.

Case:

1) If the cardinality of *ACA* is 0 (zero), then <curvepolygonm binary representation> produces an empty set of type *ST_CurvePolygon*.

2) If the cardinality of *ACA* is 1 (one), then let *AER* be the element of *ACA*. <curvepolygonm binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER)*.

3) Otherwise, let *AER* be the first element in *ACA* and let *AIR* be the sublist of *ACA* containing the other elements of *ACA*. <curvepolygonm binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER, AIR)*.

ii) If <curvepolygonm binary representation> immediately contains a <polygonm binary representation>, then <curvepolygonm binary representation> produces an *ST_Polygon* value specified by the immediately contained <polygonm binary representation>.

iii) Otherwise, <curvepolygonm binary representation> produces an empty set of type *ST_CurvePolygon*.

bw) Case:

i) If <curvepolygon binary representation> immediately contains a <num>, then <curvepolygon binary representation> produces an *ST_CurvePolygon*. Let *ACA* be an *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <wkbring binary>s.

Case:

1) If the cardinality of *ACA* is 0 (zero), then <curvepolygon binary representation> produces an empty set of type *ST_CurvePolygon*.

2) If the cardinality of *ACA* is 1 (one), then let *AER* be the element of *ACA*. <curvepolygon binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER)*.

3) Otherwise, let *AER* be the first element in *ACA* and let *AIR* be the sublist of *ACA* containing the other elements of *ACA*. <curvepolygon binary representation> produces an *ST_CurvePolygon* value as the result of the value expression: *NEW ST_CurvePolygon(AER, AIR)*.

ii) If <curvepolygon binary representation> immediately contains a <polygon binary representation>, then <curvepolygon binary representation> produces an *ST_Polygon* value specified by the immediately contained <polygon binary representation>.

iii) Otherwise, <curvepolygon binary representation> produces an empty set of type *ST_CurvePolygon*.

bx) Case:

i) If <polygonzm binary representation> immediately contains <num>, then <polygonzm binary representation> is the well-known binary representation for an *ST_Polygon* value. Let *ALSA* be an *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <wkblinearringzm binary>s.

Case:

1) If the cardinality of *ALSA* is 0 (zero), then <polygonzm binary representation> produces an empty set of type *ST_Polygon*.

2) If the cardinality of *ALSA* is 1 (one), then let *ALS* be the element of *ALSA*. <polygonzm binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(ALS)*.

3) Otherwise, let *AER* be the first element in *ALSA* and let *AIR* be the sublist of *ALSA* containing the other elements of *ALSA*. <polygonzm binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(AER, AIR)*.

ii) If <polygonzm binary representation> immediately contains a <trianglezm binary representation>, then <polygonzm binary representation> produces an *ST_Triangle* value specified by the immediately contained <trianglezm binary representation>.

iii) Otherwise, <polygonzm binary representation> produces an empty set of type *ST_Polygon*.

by) Case:

i) If <polygonz binary representation> immediately contains <num>, then <polygonz binary representation> is the well-known binary representation for an *ST_Polygon* value. Let *ALSA* be an *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <wkblinearringz binary>s.

Case:

1) If the cardinality of *ALSA* is 0 (zero), then <polygonz binary representation> produces an empty set of type *ST_Polygon*.

2) If the cardinality of *ALSA* is 1 (one), then let *ALS* be the element of *ALSA*. <polygonz binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(ALS)*.

3) Otherwise, let *AER* be the first element in *ALSA* and let *AIR* be the sublist of *ALSA* containing the other elements of *ALSA*. <polygonz binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(AER, AIR)*.

ii) If <polygonz binary representation> immediately contains a <trianglez binary representation>, then <polygonz binary representation> produces an *ST_Triangle* value specified by the immediately contained <trianglez binary representation>.

iii) Otherwise, <polygonz binary representation> produces an empty set of type *ST_Polygon*.

bz) Case:

i) If <polygonm binary representation> immediately contains <num>, then <polygonm binary representation> is the well-known binary representation for an *ST_Polygon* value. Let *ALSA* be an *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <wkblinearringm binary>s.

Case:

1) If the cardinality of *ALSA* is 0 (zero), then <polygonm binary representation> produces an empty set of type *ST_Polygon*.

2) If the cardinality of *ALSA* is 1 (one), then let *ALS* be the element of *ALSA*. <polygonm binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(ALS)*.

3) Otherwise, let *AER* be the first element in *ALSA* and let *AIR* be the sublist of *ALSA* containing the other elements of *ALSA*. <polygonm binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(AER, AIR)*.

ii) If <polygonm binary representation> immediately contains a <trianglem binary representation>, then <polygonm binary representation> produces an *ST_Triangle* value specified by the immediately contained <trianglem binary representation>.

iii) Otherwise, <polygonm binary representation> produces an empty set of type *ST_Polygon*.

ca) Case:

i) If <polygon binary representation> immediately contains <num>, then <polygon binary representation> is the well-known binary representation for an *ST_Polygon* value. Let *ALSA* be an *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <wkblinearring binary>s.

Case:

1) If the cardinality of *ALSA* is 0 (zero), then <polygon binary representation> produces an empty set of type *ST_Polygon*.

2) If the cardinality of *ALSA* is 1 (one), then let *ALS* be the element of *ALSA*. <polygon binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon(ALS)*.

3) Otherwise, let *AER* be the first element in *ALSA* and let *AIR* be the sublist of *ALSA* containing the other elements of *ALSA*. <polygon binary representation> produces an *ST_Polygon* value as the result of the value expression: *NEW ST_Polygon*(*AER, AIR*).

ii) If <polygon binary representation> immediately contains a <triangle binary representation>, then <polygon binary representation> produces an *ST_Triangle* value specified by the immediately contained <triangle binary representation>.

iii) Otherwise, <polygon binary representation> produces an empty set of type *ST_Polygon*.

cb) Case:

i) If <trianglezm binary representation> immediately contains <wkbpointzm binary>, then <trianglezm binary representation> is the well-known binary representation for an *ST_Triangle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of three that contains the *ST_Point* values specified by the immediately contained <wkbpointzm binary>s. Then <trianglezm binary representation>> produces an *ST_Triangle* value as the result of the value expression*: NEW ST_Triangle(APA)*.

ii) Otherwise, <trianglezm binary representation> produces an empty set of type *ST_Triangle*.

cc) Case:

i) If <trianglez binary representation> immediately contains <wkbpointz binary>, then <trianglez binary representation> is the well-known binary representation for an *ST_Triangle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of three that contains the *ST_Point* values specified by the immediately contained <wkbpointz binary>s. Then <trianglez binary representation>> produces an *ST_Triangle* value as the result of the value expression: *NEW ST_Triangle(APA)*.

ii) Otherwise, <trianglez binary representation> produces an empty set of type *ST_Triangle*.

cd) Case:

i) If <trianglem binary representation> immediately contains <wkbpointm binary>, then <trianglem binary representation> is the well-known binary representation for an *ST_Triangle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of three that contains the *ST_Point* values specified by the immediately contained <wkbpointm binary>s. Then <trianglem binary representation>> produces an *ST_Triangle* value as the result of the value expression: *NEW ST_Triangle(APA)*.

ii) Otherwise, <trianglem binary representation> produces an empty set of type *ST_Triangle*.

ce) Case:

i) If <triangle binary representation> immediately contains three <wkbpoint binary>s, then <triangle binary representation> is the well-known binary representation for an *ST_Triangle* value. Let *APA* be an *ST_Point* ARRAY value with cardinality of three that contains the *ST_Point* values specified by the three immediately contained <wkbpoint binary>s. Then <triangle binary representation>> produces an *ST_Triangle* value as the result of the value expression: *NEW ST_Triangle(APA)*).

ii) Otherwise, <triangle binary representation> produces an empty set of type *ST_Triangle*.

cf) Case:

i) If <polyhedralsurfacezm binary representation> immediately contains <num>, then <polyhedralsurfacezm binary representation> is the well-known binary representation for an ST_*PolyhdrlSurface* value. Let *APA* be an *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <wkbpolygonpatchzm binary>s.

Case:

1) If the cardinality of *APA* is 0 (zero), then <polyhedralsurfacezm binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

2) Otherwise, <polyhedralsurfacezm binary representation> produces an ST_*PolyhdrlSurface* value as the result of the value expression: *NEW ST_PolyhdrlSurface(APA)*.

ii) If <polyhedralsurfacezm binary representation> immediately contains a <tinzm binary representation>, then <polyhedralsurfacezm binary representation> produces an *ST_TIN* value specified by the immediately contained <tinzm binary representation>.

iii) Otherwise, <polyhedralsurfacezm binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

cg) Case:

i) If <polyhedralsurfacez binary representation> immediately contains <num>, then <polyhedralsurfacez binary representation> is the well-known binary representation for an ST_*PolyhdrlSurface* value. Let *APA* be an *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <wkbpolygonpatchz binary>s.

Case:

1) If the cardinality of *APA* is 0 (zero), then <polyhedralsurfacez binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

2) Otherwise, <polyhedralsurfacez binary representation> produces an ST_*PolyhdrlSurface* value as the result of the value expression: *NEW ST_PolyhdrlSurface(APA)*.

ii) If <polyhedralsurfacez binary representation> immediately contains a <tinz binary representation>, then <polyhedralsurfacez binary representation> produces an *ST_TIN* value specified by the immediately contained <tinz binary representation>.

iii) Otherwise, <polyhedralsurfacez binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

ch) Case:

i) If <polyhedralsurfacem binary representation> immediately contains <num>, then <polyhedralsurfacem binary representation> is the well-known binary representation for an ST_*PolyhdrlSurface* value. Let *APA* be an *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <wkbpolygonpatchm binary>s.

Case:

1) If the cardinality of *APA* is 0 (zero), then <polyhedralsurfacem binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

2) Otherwise, <polyhedralsurfacem binary representation> produces an ST_*PolyhdrlSurface* value as the result of the value expression: *NEW ST_PolyhdrlSurface(APA)*.

ii) If <polyhedralsurfacem binary representation> immediately contains a <tinm binary representation>, then <polyhedralsurfacem binary representation> produces an *ST_TIN* value specified by the immediately contained <tinm binary representation>.

iii) Otherwise, <polyhedralsurfacem binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

ci) Case:

i) If <polyhedralsurface binary representation> immediately contains <num>, then <polyhedralsurface binary representation> is the well-known binary representation for an ST_*PolyhdrlSurface* value. Let *APA* be an *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <wkbpolygonpatch binary>s.

Case:

1) If the cardinality of *APA* is 0 (zero), then <polyhedralsurface binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

2) Otherwise, <polyhedralsurface binary representation> produces an ST_*PolyhdrlSurface* value as the result of the value expression: *NEW ST_PolyhdrlSurface(APA)*.

ii) If <polyhedralsurface binary representation> immediately contains a <tin binary representation>, then <polyhedralsurface binary representation> produces an *ST_TIN* value specified by the immediately contained <tin binary representation>.

iii) Otherwise, <polyhedralsurface binary representation> produces an empty set of type ST_*PolyhdrlSurface*.

cj) Case:

i) If <tinzm binary representation> immediately contains <num>, then <tinzm binary representation> is the well-known binary representation for an *ST_TIN* value. Let *ATA* be an *ST_Triangle* ARRAY value with cardinality of <num> that contains the *ST_Triangle* values specified by the immediately contained <wkbtrianglepatchzm binary>s. Let *AEA* be an *ST_TINElement* ARRAY value with cardinality of <nume> that contains the *ST_TINElement* values specified by the immediately contained <wkbtinelement binary>s. Let *M* be a DOUBLE PRECISION value specified by the immediately contained <wkbmaxsidelength>.

Case:

1) If the cardinality of *ATA* is 0 (zero), then <tinzm binary representation> produces an empty set of type *ST_TIN*.

2) Otherwise, <tinzm binary representation> produces an *ST_TIN* value as the result of the value expression: *NEW ST_TIN(APA, AEA, M)*.

ii) Otherwise, <tinzm binary representation> produces an empty set of type *ST_TIN*.

ck) Case:

i) If <tinz binary representation> immediately contains <num>, then <tinz binary representation> is the well-known binary representation for an *ST_TIN* value. Let *ATA* be an *ST_Triangle* ARRAY value with cardinality of <num> that contains the *ST_Triangle* values specified by the immediately contained <wkbtrianglepatchz binary>s. Let *AEA* be an *ST_TINElement* ARRAY value with cardinality of <nume> that contains the *ST_TINElement* values specified by the immediately contained <wkbtinelement binary>s. Let *M* be a DOUBLE PRECISION value specified by the immediately contained <wkbmaxsidelength>.

Case:

1) If the cardinality of *ATA* is 0 (zero), then <tinz binary representation> produces an empty set of type *ST_TIN*.

2) Otherwise, <tinz binary representation> produces an *ST_TIN* value as the result of the value expression: *NEW ST_TIN(APA, AEA, M)*.

ii) Otherwise, <tinz binary representation> produces an empty set of type *ST_TIN*.

cl) Case:

i) If <tinm binary representation> immediately contains <num>, then <tinm binary representation> is the well-known binary representation for an *ST_TIN* value. Let *ATA* be an *ST_Triangle* ARRAY value with cardinality of <num> that contains the *ST_Triangle* values specified by the immediately contained <wkbtrianglepatchm binary>s. Let *AEA* be an *ST_TINElement* ARRAY value with cardinality of <nume> that contains the *ST_TINElement* values specified by the immediately contained <wkbtinelement binary>s. Let *M* be a DOUBLE PRECISION value specified by the immediately contained <wkbmaxsidelength>.

Case:

1) If the cardinality of *ATA* is 0 (zero), then <tinm binary representation> produces an empty set of type *ST_TIN*.

2) Otherwise, <tinm binary representation> produces an *ST_TIN* value as the result of the value expression: *NEW ST_TIN(APA, AEA, M)*.

ii) Otherwise, <tinm binary representation> produces an empty set of type *ST_TIN*.

cm) Case:

i) If <tin binary representation> immediately contains <num>, then <tin binary representation> is the well-known binary representation for an *ST_TIN* value. Let *ATA* be an *ST_Triangle* ARRAY value with cardinality of <num> that contains the *ST_Triangle* values specified by the immediately contained <wkbtrianglepatch binary>s. Let *AEA* be an *ST_TINElement* ARRAY value with cardinality of <nume> that contains the *ST_TINElement* values specified by the immediately contained <wkbtinelement binary>s. Let *M* be a DOUBLE PRECISION value specified by the immediately contained <wkbmaxsidelength>.

Case:

1) If the cardinality of *ATA* is 0 (zero), then <tin binary representation> produces an empty set of type *ST_TIN*.

2) Otherwise, <tin binary representation> produces an *ST_TIN* value as the result of the value expression: *NEW ST_TIN(APA, AEA, M)*.

ii) Otherwise, <tin binary representation> produces an empty set of type ST_TIN.

cn) Let *TET* be the value of <tinelement element type> in <tinelement binary representation>, let *TEID* be the value of <tinelement element id>, let *TETAG* be the value of <tinelement element tag> and let *TEGEOM* be the value of <well-known binary representation>. <tinelement binary representation> produces an *ST_TINElement* value as the result of the value expression: *NEW ST_TINElement(TET, TEID, TETAG, TEGEOM)*.

co) <tinelement element type> produces a CHARACTER VARYING(30) value of length <byte> containing the characters in <letters>.

cp) <tinelement element id> produces an INTEGER value of <signed integer>.

cq) <tinelement element tag> produces a CHARACTER VARYING(64) value of length <byte> containing the characters in <letters>.

cr) Case:

i) If <compoundsurfacezm binary representation> immediately contains <num>, then <compoundsurfacezm binary representation> is the well-known binary representation for an *ST_CompoundSurface* value. Let *ACA* be an *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <wkbsurfacezm binary representation>s. <compoundsurfacezm binary representation> produces an *ST_CompoundSurface* value as the result of the value expression: *NEW ST_CompoundSurface(ACA)*.

ii) Otherwise, <compoundsurfacezm binary representation> produces an empty set of type *ST_CompoundSurface*.

cs) Case:

i) If <compoundsurfacez binary representation> immediately contains <num>, then <compoundsurfacez binary representation> is the well-known binary representation for an *ST_CompoundSurface* value. Let *ACA* be an *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <wkbsurfacez binary representation>s. <compoundsurfacez binary representation> produces an *ST_CompoundSurface* value as the result of the value expression: *NEW ST_CompoundSurface(ACA)*.

ii) Otherwise, <compoundsurfacez binary representation> produces an empty set of type *ST_CompoundSurface*.

ct) Case:

i) If <compoundsurfacem binary representation> immediately contains <num>, then <compoundsurfacem binary representation> is the well-known binary representation for an *ST_CompoundSurface* value. Let *ACA* be an *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <wkbsurfacem binary representation>s. <compoundsurfacem binary representation> produces an *ST_CompoundSurface* value as the result of the value expression: *NEW ST_CompoundSurface(ACA)*.

ii) Otherwise, <compoundsurfacem binary representation> produces an empty set of type *ST_CompoundSurface*.

cu) Case:

i) If <compoundsurface binary representation> immediately contains <num>, then <compoundsurface binary representation> is the well-known binary representation for an *ST_CompoundSurface* value. Let *ACA* be an *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <wkbsurface binary representation>s. <compoundsurface binary representation> produces an *ST_CompoundSurface* value as the result of the value expression: *NEW ST_CompoundSurface(ACA)*.

ii) Otherwise, <compoundsurface binary representation> produces an empty set of type *ST_CompoundSurface*.

cv) <solidz binary representation> produces an *ST_BRepSolid* value specified by the immediately contained <brepsolidz binary representation>.

cw) Case:

i) If <brepsolidz binary representation> immediately contains <num>, then <brepsolidz binary representation> is the well-known binary representation for an *ST_BRepSolid* value. Let *ASA* be an *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <wkbshellz binary>s.

Case:

1) If the cardinality of *ASA* is 0 (zero), then <brepsolidz binary representation> produces an empty set of type *ST_BRepSolid*.

2) If the cardinality of *ASA* is 1 (one), then let *AES* be the element of *ASA*. <brepsolidz binary representation> produces an *ST_BRepSolid* value as the result of the value expression: *NEW ST_BRepSolid*(*AES*).

3) Otherwise, let *AES* be the first element in *ASA* and let *AIS* be the sublist of *ASA* containing the other elements of *ASA*. <brepsolidz binary representation> produces an *ST_BRepSolid* value as the result of the value expression: *NEW ST_BRepSolid*(*AES*, *AIS*).

ii) Otherwise, <brepsolidz binary representation> produces an empty set of type *ST_BRepSolid*.

cx) Case:

i) If <collectionzm binary representation> immediately contains a <multipointzm binary representation>, then <collectionzm binary representation> produces an *ST_MultiPoint* value specified by the immediately contained <multipointzm binary representation>.

ii) If <collectionzm binary representation> immediately contains a <multicurvezm binary representation>, then <collectionzm binary representation> produces an *ST_MultiCurve* value specified by the immediately contained <multicurvezm binary representation>.

iii) If <collectionzm binary representation> immediately contains a <multisurfacezm binary representation>, then <collectionzm binary representation> produces an *ST_MultiSurface* value specified by the immediately contained <multisurfacezm binary representation>.

iv) Otherwise, <collectionzm binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <geometrycollection binary representation>.

cy) Case:

i) If <collectionz binary representation> immediately contains a <multipointz binary representation>, then <collectionz binary representation> produces an *ST_MultiPoint* value specified by the immediately contained <multipointz binary representation>.

ii) If <collectionz binary representation> immediately contains a <multicurvez binary representation>, then <collectionz binary representation> produces an *ST_MultiCurve* value specified by the immediately contained <multicurvez binary representation>.

iii) If <collectionz binary representation> immediately contains a <multisurfacez binary representation>, then <collectionz binary representation> produces an *ST_MultiSurface* value specified by the immediately contained <multisurfacez binary representation>.

iv) Otherwise, <collectionz binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <geometrycollection binary representation>.

cz) Case:

i) If <collectionm binary representation> immediately contains a <multipointm binary representation>, then <collectionm binary representation> produces an *ST_MultiPoint* value specified by the immediately contained <multipointm binary representation>.

ii) If <collectionm binary representation> immediately contains a <multicurvem binary representation>, then <collectionm binary representation> produces an *ST_MultiCurve* value specified by the immediately contained <multicurvem binary representation>.

iii) If <collectionm binary representation> immediately contains a <multisurfacem binary representation>, then <collectionm binary representation> produces an *ST_MultiSurface* value specified by the immediately contained <multisurfacem binary representation>.

iv) Otherwise, <collectionm binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <geometrycollection binary representation>.

da) Case:

i) If <collection binary representation> immediately contains a <multipoint binary representation>, then <collection binary representation> produces an *ST_MultiPoint* value specified by the immediately contained <multipoint binary representation>.

ii) If <collection binary representation> immediately contains a <multicurve binary representation>, then <collection binary representation> produces an *ST_MultiCurve* value specified by the immediately contained <multicurve binary representation>.

iii) If <collection binary representation> immediately contains a <multisurface binary representation>, then <collection binary representation> produces an *ST_MultiSurface* value specified by the immediately contained <multisurface binary representation>.

iv) Otherwise, <collection binary representation> produces an *ST_GeomCollection* value specified by the immediately contained <geometrycollection binary representation>.

db) Case:

i) If <multipointzm binary representation> immediately contains <num>, then <multipointzm binary representation> is the well-known binary representation for an *ST_MultiPoint* value. Let *APA* be the *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <pointzm binary representation>s. <multipointzm binary representation> produces an *ST_MultiPoint* value as the result of the value expression: *NEW ST_MultiPoint(APA)*.

ii) Otherwise, <multipointzm binary representation> produces an empty set of type *ST_MultiPoint*.

dc) Case:

i) If <multipointz binary representation> immediately contains <num>, then <multipointz binary representation> is the well-known binary representation for an *ST_MultiPoint* value. Let *APA* be the *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <pointz binary representation>s. <multipointz binary representation> produces an *ST_MultiPoint* value as the result of the value expression: *NEW ST_MultiPoint(APA)*.

ii) Otherwise, <multipointz binary representation> produces an empty set of type *ST_MultiPoint*.

dd) Case:

i) If <multipointm binary representation> immediately contains <num>, then <multipointm binary representation> is the well-known binary representation for an *ST_MultiPoint* value. Let *APA* be the *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <pointm binary representation>s. <multipointm binary representation> produces an *ST_MultiPoint* value as the result of the value expression: *NEW ST_MultiPoint(APA)*.

ii) Otherwise, <multipointm binary representation> produces an empty set of type *ST_MultiPoint*.

de) Case:

i) If <multipoint binary representation> immediately contains <num>, then <multipoint binary representation> is the well-known binary representation for an *ST_MultiPoint* value. Let *APA* be the *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <point binary representation>s. <multipoint binary representation> produces an *ST_MultiPoint* value as the result of the value expression: *NEW ST_MultiPoint(APA)*.

ii) Otherwise, <multipoint binary representation> produces an empty set of type *ST_MultiPoint*.

df) Case:

i) If <multicurvezm binary representation> immediately contains a <num>, then <multicurvezm binary representation> produces an *ST_MultiCurve* value. Let *ACA* be the *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <curvezm binary representation>s. <multicurvezm binary representation> produces an *ST_MultiCurve* value as the result of the value expression: *NEW ST_MultiCurve(ACA)*.

ii) If <multicurvezm binary representation> immediately contains a <multilinestringzm binary representation>, then <multicurvezm binary representation> produces an *ST_MultiLineString* value specified by the immediately contained <multilinestringzm binary representation>.

iii) Otherwise, <multicurvezm binary representation> produces an empty set of type *ST_MultiCurve*.

dg) Case:

i) If <multicurvez binary representation> immediately contains a <num>, then <multicurvez binary representation> produces an *ST_MultiCurve* value. Let *ACA* be the *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <curvez binary representation>s. <multicurvez binary representation> produces an *ST_MultiCurve* value as the result of the value expression: *NEW ST_MultiCurve(ACA)*.

ii) If <multicurvez binary representation> immediately contains a <multilinestringz binary representation>, then <multicurvez binary representation> produces an *ST_MultiLineString* value specified by the immediately contained <multilinestringz binary representation>.

iii) Otherwise, <multicurvez binary representation> produces an empty set of type *ST_MultiCurve*.

dh) Case:

i) If <multicurvem binary representation> immediately contains a <num>, then <multicurvem binary representation> produces an *ST_MultiCurve* value. Let *ACA* be the *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <curvem binary representation>s. <multicurvem binary representation> produces an *ST_MultiCurve* value as the result of the value expression: *NEW ST_MultiCurve(ACA)*.

ii) If <multicurvem binary representation> immediately contains a <multilinestringm binary representation>, then <multicurvem binary representation> produces an *ST_MultiLineString* value specified by the immediately contained <multilinestringm binary representation>.

iii) Otherwise, <multicurvem binary representation> produces an empty set of type *ST_MultiCurve*.

di) Case:

i) If <multicurve binary representation> immediately contains a <num>, then <multicurve binary representation> produces an *ST_MultiCurve* value. Let *ACA* be the *ST_Curve* ARRAY value with cardinality of <num> that contains the *ST_Curve* values specified by the immediately contained <curve binary representation>s. <multicurve binary representation> produces an *ST_MultiCurve* value as the result of the value expression: *NEW ST_MultiCurve(ACA)*.

ii) If <multicurve binary representation> immediately contains a <multilinestring binary representation>, then <multicurve binary representation> produces an *ST_MultiLineString* value specified by the immediately contained <multilinestring binary representation>.

iii) Otherwise, <multicurve binary representation> produces an empty set of type *ST_MultiCurve*.

dj) Case:

i) If <multilinestringzm binary representation> immediately contains <num>, then <multilinestringzm binary representation> is the well-known binary representation for an *ST_MultiLineString* value. Let *ALSA* be the *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <linestringzm binary representation>s. <multilinestringzm binary representation> produces an *ST_MultiLineString* value as the result of the value expression: *NEW ST_MultiLineString(ALSA)*.

ii) Otherwise, <multilinestringzm binary representation> produces an empty set of type *ST_MultiLineString*.

dk) Case:

i) If <multilinestringz binary representation> immediately contains <num>, then <multilinestringz binary representation> is the well-known binary representation for an *ST_MultiLineString* value. Let *ALSA* be the *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <linestringz binary representation>s. <multilinestringz binary representation> produces an *ST_MultiLineString* value as the result of the value expression: *NEW ST_MultiLineString(ALSA)*.

ii) Otherwise, <multilinestringz binary representation> produces an empty set of type *ST_MultiLineString*.

dl) Case:

i) If <multilinestringm binary representation> immediately contains <num>, then <multilinestringm binary representation> is the well-known binary representation for an *ST_MultiLineString* value. Let *ALSA* be the *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <linestringm binary representation>s. <multilinestringm binary representation> produces an *ST_MultiLineString* value as the result of the value expression: *NEW ST_MultiLineString(ALSA)*.

ii) Otherwise, <multilinestringm binary representation> produces an empty set of type *ST_MultiLineString*.

dm) Case:

i) If <multilinestring binary representation> immediately contains <num>, then <multilinestring binary representation> is the well-known binary representation for an *ST_MultiLineString* value. Let *ALSA* be the *ST_LineString* ARRAY value with cardinality of <num> that contains the *ST_LineString* values specified by the immediately contained <linestring binary representation>s. <multilinestring binary representation> produces an *ST_MultiLineString* value as the result of the value expression: *NEW ST_MultiLineString(ALSA)*.

ii) Otherwise, <multilinestring binary representation> produces an empty set of type *ST_MultiLineString*.

dn) Case:

i) If <multisurfacezm binary representation> immediately contains a <num>, then <multisurfacezm binary representation> produces an *ST_MultiSurface* value. Let *ASA* be the *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <surfacezm binary representation>s. <multisurfacezm binary representation> produces an *ST_MultiSurface* value as the result of the value expression: *NEW ST_MultiSurface(ASA)*.

ii) If <multisurfacezm binary representation> immediately contains a <multipolygonzm binary representation>, then <multisurfacezm binary representation> produces an *ST_MultiPolygon* value specified by the immediately contained <multipolygonzm binary representation>.

iii) Otherwise, <multisurfacezm binary representation> produces an empty set of type *ST_MultiSurface*.

do) Case:

i) If <multisurfacez binary representation> immediately contains a <num>, then <multisurfacez binary representation> produces an *ST_MultiSurface* value. Let *ASA* be the *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <surfacez binary representation>s. <multisurfacez binary representation> produces an *ST_MultiSurface* value as the result of the value expression: *NEW ST_MultiSurface(ASA)*.

ii) If <multisurfacez binary representation> immediately contains a <multipolygonz binary representation>, then <multisurfacez binary representation> produces an *ST_MultiPolygon* value specified by the immediately contained <multipolygonz binary representation>.

iii) Otherwise, <multisurfacez binary representation> produces an empty set of type *ST_MultiSurface*.

dp) Case:

i) If <multisurfacem binary representation> immediately contains a <num>, then <multisurfacem binary representation> produces an *ST_MultiSurface* value. Let *ASA* be the *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <surfacem binary representation>s. <multisurfacem binary representation> produces an *ST_MultiSurface* value as the result of the value expression: *NEW ST_MultiSurface(ASA)*.

ii) If <multisurfacem binary representation> immediately contains a <multipolygonm binary representation>, then <multisurfacem binary representation> produces an *ST_MultiPolygon* value specified by the immediately contained <multipolygonm binary representation>.

iii) Otherwise, <multisurfacem binary representation> produces an empty set of type *ST_MultiSurface*.

dq) Case:

i) If <multisurface binary representation> immediately contains a <num>, then <multisurface binary representation> produces an *ST_MultiSurface* value. Let *ASA* be the *ST_Surface* ARRAY value with cardinality of <num> that contains the *ST_Surface* values specified by the immediately contained <surface binary representation>s. <multisurface binary representation> produces an *ST_MultiSurface* value as the result of the value expression: *NEW ST_MultiSurface(ASA)*.

ii) If <multisurface binary representation> immediately contains a <multipolygon binary representation>, then <multisurface binary representation> produces an *ST_MultiPolygon* value specified by the immediately contained <multipolygon binary representation>.

iii) Otherwise, <multisurface binary representation> produces an empty set of type *ST_MultiSurface*.

dr) Case:

i) If <multipolygonzm binary representation> immediately contains <num>, then <multipolygonzm binary representation> is the well-known binary representation for an *ST_MultiPolygon* value. Let *APA* be the *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <polygonzm binary representation>s. <multipolygonzm binary representation> produces an *ST_MultiPolygon* value as the result of the value expression: *NEW ST_MultiPolygon(APA)*.

ii) Otherwise, <multipolygonzm binary representation> produces an empty set of type *ST_MultiPolygon*.

ds) Case:

i) If <multipolygonz binary representation> immediately contains <num>, then <multipolygonz binary representation> is the well-known binary representation for an *ST_MultiPolygon* value. Let *APA* be the *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <polygonz binary representation>s. <multipolygonz binary representation> produces an *ST_MultiPolygon* value as the result of the value expression: *NEW ST_MultiPolygon*(*APA*).

ii) Otherwise, <multipolygonz binary representation> produces an empty set of type *ST_MultiPolygon*.

dt) Case:

i) If <multipolygonm binary representation> immediately contains <num>, then <multipolygonm binary representation> is the well-known binary representation for an *ST_MultiPolygon* value. Let *APA* be the *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <polygonm binary representation>s. <multipolygonm binary representation> produces an *ST_MultiPolygon* value as the result of the value expression: *NEW ST_MultiPolygon(APA)*.

ii) Otherwise, <multipolygonm binary representation> produces an empty set of type *ST_MultiPolygon*.

du) Case:

i) If <multipolygon binary representation> immediately contains <num>, then <multipolygon binary representation> is the well-known binary representation for an *ST_MultiPolygon* value. Let *APA* be the *ST_Polygon* ARRAY value with cardinality of <num> that contains the *ST_Polygon* values specified by the immediately contained <polygon binary representation>s. <multipolygon binary representation> produces an *ST_MultiPolygon* value as the result of the value expression: *NEW ST_MultiPolygon(APA)*.

ii) Otherwise, <multipolygon binary representation> produces an empty set of type *ST_MultiPolygon*.

dv) Case:

i) If <geometrycollectionzm binary representation> immediately contains <num>, then <geometrycollectionzm binary representation> is the well-known binary representation for an *ST_GeomCollection*. Let *AGA* be the *ST_Geometry* ARRAY value with cardinality of <num> that contains the *ST_Geometry* values specified by the immediately contained <well-knownzm binary representation>s. <geometrycollectionzm binary representation> produces an *ST_GeomCollection* value as the result of the value expression: *NEW ST_GeomCollection(AGA)*.

ii) Otherwise, <geometrycollectionzm binary representation> produces an empty set of type *ST_GeomCollection*.

dw) Case:

i) If <geometrycollectionz binary representation> immediately contains <num>, then <geometrycollectionz binary representation> is the well-known binary representation for an *ST_GeomCollection*. Let *AGA* be the *ST_Geometry* ARRAY value with cardinality of <num> that contains the *ST_Geometry* values specified by the immediately contained <well-knownz binary representation>s. <geometrycollectionz binary representation> produces an *ST_GeomCollection* value as the result of the value expression: *NEW ST_GeomCollection(AGA)*.

ii) Otherwise, <geometrycollectionz binary representation> produces an empty set of type *ST_GeomCollection*.

dx) Case:

i) If <geometrycollectionm binary representation> immediately contains <num>, then <geometrycollectionm binary representation> is the well-known binary representation for an *ST_GeomCollection*. Let *AGA* be the *ST_Geometry* ARRAY value with cardinality of <num> that contains the *ST_Geometry* values specified by the immediately contained <well-known binarym representation>s. <geometrycollectionm binary representation> produces an *ST_GeomCollection* value as the result of the value expression: *NEW ST_GeomCollection(AGA)*.

ii) Otherwise, <geometrycollectionm binary representation> produces an empty set of type *ST_GeomCollection*.

dy) Case:

i) If <geometrycollection binary representation> immediately contains <num>, then <geometrycollection binary representation> is the well-known binary representation for an *ST_GeomCollection*. Let *AGA* be the *ST_Geometry* ARRAY value with cardinality of <num> that contains the *ST_Geometry* values specified by the immediately contained <well-known2d binary representation>s. <geometrycollection binary representation> produces an *ST_GeomCollection* value as the result of the value expression: *NEW ST_GeomCollection(AGA)*.

ii) Otherwise, <geometrycollection binary representation> produces an empty set of type *ST_GeomCollection*.

dz) <wkbpolygonpatchzm binary> produces an *ST_Polygon* value specified by the immediately contained <polygonzm binary representation>.

ea) <wkbpolygonpatchz binary> produces an *ST_Polygon* value specified by the immediately contained <polygonz binary representation>.

eb) <wkbpolygonpatchm binary> produces an *ST_Polygon* value specified by the immediately contained <polygonm binary representation>.

ec) <wkbpolygonpatch binary> produces an *ST_Polygon* value specified by the immediately contained <polygon binary representation>.

ed) <wkbtrianglepatchzm binary> produces an *ST_Triangle* value specified by the immediately contained <trianglezm binary representation>.

ee) <wkbtrianglepatchz binary> produces an *ST_Triangle* value specified by the immediately contained <trianglez binary representation>.

ef) <wkbtrianglepatchm binary> produces an *ST_Triangle* value specified by the immediately contained <trianglem binary representation>.

eg) <wkbtrianglepatch binary> produces an *ST_Triangle* value specified by the immediately contained <triangle binary representation>.

eh) <wkbtinelement binary> produces an *ST_TINElement* value specified by the immediately contained <tinelement binary representation>.

ei) Case:

  i) If <wkbcurvezm binary> immediately contains a <linestringzm binary representation>, then <wkbcurvezm binary> produces an *ST_LineString* value specified by the immediately contained <linestringzm binary representation>.

  ii) If <wkbcurvezm binary> immediately contains a <circularstringzm binary representation>, then <wkbcurvezm binary> produces an *ST_CircularString* value specified by the immediately contained <circularstringzm binary representation>.

  iii) If <wkbcurvezm binary> immediately contains a <circlezm binary representation>, then <wkbcurvezm binary> produces an *ST_Circle* value specified by the immediately contained <circlezm binary representation>.

  iv) If <wkbcurvezm binary> immediately contains a <geodesiczm binary representation>, then <wkbcurvezm binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesiczm binary representation>.

  v) If <wkbcurvezm binary> immediately contains a <ellipticalzm binary representation>, then <wkbcurvezm binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalzm binary representation>.

  vi) If <wkbcurvezm binary> immediately contains a <nurbszm binary representation>, then <wkbcurvezm binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbszm binary representation>.

  vii) If <wkbcurvezm binary> immediately contains a <clothoidzm binary representation>, then <wkbcurvezm binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoidzm binary representation>.

  viii) If <wkbcurvezm binary> immediately contains a <spiralzm binary representation>, then <wkbcurvezm binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralzm binary representation>.

  ix) Otherwise, <wkbcurvezm binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvezm binary representation>.

ej) Case:

  i) If <wkbcurvez binary> immediately contains a <linestringz binary representation>, then <wkbcurvez binary> produces an *ST_LineString* value specified by the immediately contained <linestringz binary representation>.

  ii) If <wkbcurvez binary> immediately contains a <circularstringz binary representation>, then <wkbcurvez binary> produces an *ST_CircularString* value specified by the immediately contained <circularstringz binary representation>.

iii) If <wkbcurvez binary> immediately contains a <circlez binary representation>, then <wkbcurvez binary> produces an *ST_Circle* value specified by the immediately contained <circlez binary representation>.

iv) If <wkbcurvez binary> immediately contains a <geodesicz binary representation>, then <wkbcurvez binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesicz binary representation>.

v) If <wkbcurvez binary> immediately contains a <ellipticalz binary representation>, then <wkbcurvez binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalz binary representation>.

vi) If <wkbcurvez binary> immediately contains a <nurbsz binary representation>, then <wkbcurvez binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbsz binary representation>.

vii) If <wkbcurvez binary> immediately contains a <clothoidz binary representation>, then <wkbcurvez binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoidz binary representation>.

viii) If <wkbcurvez binary> immediately contains a <spiralz binary representation>, then <wkbcurvez binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralz binary representation>.

ix) Otherwise, <wkbcurvez binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvez binary representation>.

ek) Case:

i) If <wkbcurvem binary> immediately contains a <linestringm binary representation>, then <wkbcurvem binary> produces an *ST_LineString* value specified by the immediately contained <linestringm binary representation>.

ii) If <wkbcurvem binary> immediately contains a <circularstringm binary representation>, then <wkbcurvem binary> produces an *ST_CircularString* value specified by the immediately contained <circularstringm binary representation>.

iii) If <wkbcurvem binary> immediately contains a <circlem binary representation>, then <wkbcurvem binary> produces an *ST_Circle* value specified by the immediately contained <circlem binary representation>.

iv) If <wkbcurvem binary> immediately contains a <geodesicm binary representation>, then <wkbcurvem binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesicm binary representation>.

v) If <wkbcurvem binary> immediately contains a <ellipticalm binary representation>, then <wkbcurvem binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalm binary representation>.

vi) If <wkbcurvem binary> immediately contains a <nurbsm binary representation>, then <wkbcurvem binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbsm binary representation>.

vii) If <wkbcurvem binary> immediately contains a <clothoidm binary representation>, then <wkbcurvem binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoidm binary representation>.

viii) If <wkbcurvem binary> immediately contains a <spiralm binary representation>, then <wkbcurvem binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralm binary representation>.

ix) Otherwise, <wkbcurvem binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvem binary representation>.

el) Case:

i) If <wkbcurve binary> immediately contains a <linestring binary representation>, then <wkbcurve binary> produces an *ST_LineString* value specified by the immediately contained <linestring binary representation>.

ii) If <wkbcurve binary> immediately contains a <circularstring binary representation>, then <wkbcurve binary> produces an *ST_CircularString* value specified by the immediately contained <circularstring binary representation>.

iii) If <wkbcurve binary> immediately contains a <circle binary representation>, then <wkbcurve binary> produces an *ST_Circle* value specified by the immediately contained <circle binary representation>.

iv) If <wkbcurve binary> immediately contains a <geodesic binary representation>, then <wkbcurve binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesic binary representation>.

v) If <wkbcurve binary> immediately contains a <elliptical binary representation>, then <wkbcurve binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical binary representation>.

vi) If <wkbcurve binary> immediately contains a <nurbs binary representation>, then <wkbcurve binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs binary representation>.

vii) If <wkbcurve binary> immediately contains a <clothoid binary representation>, then <wkbcurve binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoid binary representation>.

viii) If <wkbcurve binary> immediately contains a <spiral binary representation>, then <wkbcurve binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral binary representation>.

ix) Otherwise, <wkbcurve binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurve binary representation>.

em) Case:

i) If <wkbringzm binary> immediately contains a <linestringzm binary representation>, then <wkbringzm binary> produces an *ST_LineString* value specified by the immediately contained <linestringzm binary representation>.

ii) If <wkbringzm binary> immediately contains a <circularstringzm binary representation>, then <wkbringzm binary> produces an *ST_CircularString* value specified by the immediately contained <circularstringzm binary representation>.

iii) If <wkbringzm binary> immediately contains a <circlezm binary representation>, then <wkbringzm binary> produces an *ST_Circle* value specified by the immediately contained <circlezm binary representation>.

iv) If <wkbringzm binary> immediately contains a <geodesiczm binary representation>, then <wkbringzm binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesiczm binary representation>.

v) If <wkbringzm binary> immediately contains a <ellipticalzm binary representation>, then <wkbringzm binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalzm binary representation>.

vi) If <wkbringzm binary> immediately contains a <nurbszm binary representation>, then <wkbringzm binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbszm binary representation>.

vii) If <wkbringzm binary> immediately contains a <clothoidzm binary representation>, then <wkbringzm binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoidzm binary representation>.

viii) If <wkbringzm binary> immediately contains a <spiralzm binary representation>, then <wkbringzm binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralzm binary representation>.

ix) Otherwise, <wkbringzm binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvezm binary representation>.

en) Case:

i) If <wkbringz binary> immediately contains a <linestringz binary representation>, then <wkbringz binary> produces an *ST_LineString* value specified by the immediately contained <linestringz binary representation>.

ii) If <wkbringz binary> immediately contains a <circularstringz binary representation>, then <wkbringz binary> produces an *ST_CircularString* value specified by the immediately contained <circularstringz binary representation>.

iii) If <wkbringz binary> immediately contains a <circlez binary representation>, then <wkbringz binary> produces an *ST_Circle* value specified by the immediately contained <circlez binary representation>.

iv) If <wkbringz binary> immediately contains a <geodesicz binary representation>, then <wkbringz binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesicz binary representation>.

v) If <wkbringz binary> immediately contains a <ellipticalz binary representation>, then <wkbringz binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalz binary representation>.

vi) If <wkbringz binary> immediately contains a <nurbsz binary representation>, then <wkbringz binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbsz binary representation>.

vii) If <wkbringz binary> immediately contains a <clothoidz binary representation>, then <wkbringz binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoidz binary representation>.

viii) If <wkbringz binary> immediately contains a <spiralz binary representation>, then <wkbringz binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralz binary representation>.

ix) Otherwise, <wkbringz binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvez binary representation>.

eo) Case:

i) If <wkbringm binary> immediately contains a <linestringm binary representation>, then <wkbringm binary> produces an *ST_LineString* value specified by the immediately contained <linestringm binary representation>.

ii) If <wkbringm binary> immediately contains a <circularstringm binary representation>, then <wkbringm binary> produces an *ST_CircularString* value specified by the immediately contained <circularstringm binary representation>.

iii) If <wkbringm binary> immediately contains a <circlem binary representation>, then <wkbringm binary> produces an *ST_Circle* value specified by the immediately contained <circlem binary representation>.

iv) If <wkbringm binary> immediately contains a <geodesicm binary representation>, then <wkbringm binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesicm binary representation>.

v) If <wkbringm binary> immediately contains a <ellipticalm binary representation>, then <wkbringm binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <ellipticalm binary representation>.

vi) If <wkbringm binary> immediately contains a <nurbsm binary representation>, then <wkbringm binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbsm binary representation>.

vii) If <wkbringm binary> immediately contains a <clothoidm binary representation>, then <wkbringm binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoidm binary representation>.

viii) If <wkbringm binary> immediately contains a <spiralm binary representation>, then <wkbringm binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiralm binary representation>.

ix) Otherwise, <wkbringm binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurvem binary representation>.

ep) Case:

i) If <wkbring binary> immediately contains a <linestring binary representation>, then <wkbring binary> produces an *ST_LineString* value specified by the immediately contained <linestring binary representation>.

ii) If <wkbring binary> immediately contains a <circularstring binary representation>, then <wkbring binary> produces an *ST_CircularString* value specified by the immediately contained <circularstring binary representation>.

iii) If <wkbring binary> immediately contains a <circle binary representation>, then <wkbring binary> produces an *ST_Circle* value specified by the immediately contained <circle binary representation>.

iv) If <wkbring binary> immediately contains a <geodesic binary representation>, then <wkbring binary> produces an *ST_GeodesicString* value specified by the immediately contained <geodesic binary representation>.

v) If <wkbring binary> immediately contains a <elliptical binary representation>, then <wkbring binary> produces an *ST_EllipticalCurve* value specified by the immediately contained <elliptical binary representation>.

vi) If <wkbring binary> immediately contains a <nurbs binary representation>, then <wkbring binary> produces an *ST_NURBSCurve* value specified by the immediately contained <nurbs binary representation>.

vii) If <wkbring binary> immediately contains a <clothoid binary representation>, then <wkbring binary> produces an *ST_Clothoid* value specified by the immediately contained <clothoid binary representation>.

viii) If <wkbring binary> immediately contains a <spiral binary representation>, then <wkbring binary> produces an *ST_SpiralCurve* value specified by the immediately contained <spiral binary representation>.

ix) Otherwise, <wkbring binary> produces an *ST_CompoundCurve* value specified by the immediately contained <compoundcurve binary representation>.

eq) Case:

i) If <wkbshellz binary> immediately contains a <polyhedralsurfacez binary representation>, then <wkbshellz binary> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurfacez binary representation>.

ii) If <wkbshellz binary> immediately contains a <polyhedralsurfacezm binary representation>, then <wkbshellz binary> produces an *ST_PolyhdrlSurface* value specified by the immediately contained <polyhedralsurfacezm binary representation> without m values.

iii) If <wkbshellz binary> immediately contains a <compoundsurfacez binary representation>, then <wkbshellz binary> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurfacez binary representation>.

iv) Otherwise, <wkbshellz binary> produces an *ST_CompoundSurface* value specified by the immediately contained <compoundsurfacezm binary representation> without m values.

er) Let *XC* be the DOUBLE PRECISION value specified by <wkbx> in <wkbpointzm binary>, *YC* be the DOUBLE PRECISION value specified by <wkby> in <wkbpointzm binary>, *ZC* be the DOUBLE PRECISION value specified by <wkbz> in <wkbpointzm binary>, and *MC* be the DOUBLE PRECISION value specified by <wkbm> in <wkbpointzm binary>. <wkbpointzm binary> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point*(*XC, YC, ZC, MC*).

es) Let *XC* be the DOUBLE PRECISION value specified by <wkbx> in <wkbpointz binary>, *YC* be the DOUBLE PRECISION value specified by <wkby> in <wkbpointz binary>, and *ZC* be the DOUBLE PRECISION value specified by <wkbz> in <wkbpointz binary>. <wkbpointz binary> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC, ZC*).

et) Let *XC* be the DOUBLE PRECISION value specified by <wkbx> in <wkbpointm binary>, *YC* be the DOUBLE PRECISION value specified by <wkby> in <wkbpointm binary>, and *MC* be the DOUBLE PRECISION value specified by <wkbm> in <wkbpointm binary>. <wkbpointm binary> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC, NULL, MC*).

eu) Let *XC* be the DOUBLE PRECISION value specified by <wkbx> in <wkbpoint binary> and *YC* be the DOUBLE PRECISION value specified by <wkby> in <wkbpoint binary>. <wkbpoint binary> produces an *ST_Point* value as the result of the value expression: *NEW ST_Point(XC, YC)*.

ev) <wkbx> is a <double> representing the x coordinate value of an *ST_Point* value.

ew) <wkby> is a <double> representing the y coordinate value of an *ST_Point* value.

ex) <wkbz> is a <double> representing the z coordinate value of an *ST_Point* value.

ey) <wkbm> is a <double> representing the m coordinate value of an *ST_Point* value.

ez) <num> is an <uint32> that represents the number of elements in a repeating group.

fa) <nume> is an <uint32> that represents the number of elements in a repeating group.

fb) <wkbmaxsidelength> is a <double> representing the maxsidelength value of an *ST_TIN* value.

fc) <wkblinearringzm binary> produces an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointzm binary>s.

fd) <wkblinearringz binary> produces an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointz binary>s.

fe) <wkblinearringm binary> produces an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpointm binary>s.

ff) <wkblinearring binary> produces an *ST_Point* ARRAY value with cardinality of <num> that contains the *ST_Point* values specified by the immediately contained <wkbpoint binary>s.

fg) <wkbuaxislength> is a <double> representing the uaxislength value of an *ST_EllipticalCurve* value.

fh) <wkbvaxislength> is a <double> representing the vaxislength value of an *ST_EllipticalCurve* value.

fi) <wkbstartangle> is a <double> representing the startangle value of an *ST_EllipticalCurve* value.

fj) <wkbendangle> is a <double> representing the endangle value of an *ST_EllipticalCurve* value.

fk) <wkbstartm> is a <double> representing the startm value of an *ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid* or *ST_SpiralCurve* value.

fl) <wkbendm> is a <double> representing the endm value of an *ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid* or *ST_SpiralCurve* value.

fm) <wkbdegree> is a <byte> representing the degree value of an *ST_NURBSCurve* value.

fn) <wkbweight> is a <double> representing the weight value of an *ST_NURBSPoint* value.

fo) <wkbvalue> is a <double> representing the value value of an *ST_Knot* value.

fp) <wkbmultiplicity> is a <byte> representing the multiplicity value of an *ST_Knot* value.

fq) <wkbscalefactor> is a <double> representing the scalefactor value of an *ST_Clothoid* value.

fr) <wkbstartdistance> is a <double> representing the startdistance value of an *ST_Clothoid* value.

fs) <wkbenddistance> is a <double> representing the enddistance value of an *ST_Clothoid* value.

ft) <wkbspirallength> is a <double> representing the length value of an *ST_SpiralCurve* value.

fu) <wkbstartcurvature> is a <double> representing the startcurvature value of an *ST_SpiralCurve* value.

fv) <wkbendcurvature> is a <double> representing the endcurvature value of an *ST_SpiralCurve* value.

fw) <wkbspiraltype> is <byte> <letters> representing the spiraltype value of an *ST_SpiralCurve* value.

fx) The <well-known binary representation> <uint32> values are defined in Table 15 — <well-known binary representation> <uint32> Values.

Table 15 — <well-known binary representation> <uint32> Values

<well-known binary representation>	<uint32> Value
<wkbpoint>	1 (one)
<wkblinestring>	2
<wkbpolygon>	3
<wkbmultipoint>	4
<wkbmultilinestring>	5
<wkbmultipolygon>	6
<wkbgeometrycollection>	7
<wkbcircularstring>	8 or 1000001
<wkbcompoundcurve>	9 or 1000002
<wkbcurvepolygon>	10 or 1000003
<wkbmulticurve>	11 or 1000004
<wkbmultisurface>	12 or 1000005
<wkbpolyhedralsurface>	15
<wkbtin>	16
<wkbtriangle>	17
<wkbcircle>	18
<wkbgeodesicstring>	19
<wkbellipticalcurve>	20
<wkbnurbscurve>	21
<wkbclothoid>	22
<wkbspiralcurve>	23
<wkbcompoundsurface>	24
<wkbaffineplacement>	102
<wkbpointz>	1001
<wkblinestringz>	1002
<wkbpolygonz>	1003
<wkbmultipointz>	1004
<wkbmultilinestringz>	1005
<wkbmultipolygonz>	1006
<wkbgeometrycollectionz>	1007
<wkbcircularstringz>	1008
<wkbcompoundcurvez>	1009
<wkbcurvepolygonz>	1010
<wkbmulticurvez>	1011
<wkbmultisurfacez>	1012
<wkbpolyhedralsurfacez>	1015
<wkbtinz>	1016
<wkbtrianglez>	1017
<wkbcirclez>	1018
<wkbgeodesicstringz>	1019
<wkbellipticalcurvez>	1020
<wkbnurbscurvez>	1021
<wkbclothoidz>	1022
<wkbspiralcurvez>	1023
<wkbcompoundsurfacez>	1024
<wkbbrepsolidz>	1025
<wkbaffineplacementz>	1102
<wkbpointm>	2001
<wkblinestringm>	2002
<wkbpolygonm>	2003
<wkbmultipointm>	2004

<well-known binary representation>	<uint32> Value
<wkbmultilinestringm>	2005
<wkbmultipolygonm>	2006
<wkbgeometrycollectionm>	2007
<wkbcircularstringm>	2008
<wkbcompoundcurvem>	2009
<wkbcurvepolygonm>	2010
<wkbmulticurvem>	2011
<wkbmultisurfacem>	2012
<wkbpolyhedralsurfacem>	2015
<wkbtinm>	2016
<wkbtrianglem>	2017
<wkbcirclem>	2018
<wkbgeodesicstringm>	2019
<wkbellipticalcurvem>	2020
<wkbnurbscurvem>	2021
<wkbclothoidm>	2022
<wkbspiralcurvem>	2023
<wkbcompoundsurfacem>	2024
<wkbpointzm>	3001
<wkblinestringzm>	3002
<wkbpolygonzm>	3003
<wkbmultipointzm>	3004
<wkbmultilinestringzm>	3005
<wkbmultipolygonzm>	3006
<wkbgeometrycollectionzm>	3007
<wkbcircularstringzm>	3008
<wkbcompoundcurvezm>	3009
<wkbcurvepolygonzm>	3010
<wkbmulticurvezm>	3011
<wkbmultisurfacezm>	3012
<wkbpolyhedralsurfacezm>	3015
<wkbtinzm>	3016
<wkbtrianglezm>	3017
<wkbcirclezm>	3018
<wkbgeodesicstringzm>	3019
<wkbellipticalcurvezm>	3020
<wkbnurbscurvezm>	3021
<wkbclothoidzm>	3022
<wkbspiralcurvezm>	3023
<wkbcompoundsurfacezm>	3024

fy) <byte order> indicates the binary representation of <uint32> and <double> values that follow <byte order>.

fz) <big endian> is a <byte order> represented by a <byte> with the value 0 (zero). <uint32> is Big Endian (most significant octet first). <double> is Big Endian (sign bit is in the first octet).

ga) <little endian> is a <byte order> represented by a <byte> with the value 1 (one). <uint32> is Little Endian (most significant octet last). <double> is Little Endian (sign bit is in the last octet).

gb) <byte> is an 8 bit (1 (one) octet) data type that encodes an unsigned integer in the range [0, 255].

gc) <uint32> s a 32 bit (4 octets) data type that encodes an unsigned integer in the range [0, 4294967295].

gd) <double> is a 64 bit (8 octets) double precision data type that encodes a double precision format using the IEC 559:1989.

ge) <bit> is a single bit data type that encodes a value of 0 (zero) or 1 (one).

gf) <well-known binary representation> provides a portable representation of a geometry value as a contiguous stream of octets in a BINARY LARGE OBJECT value. The serialized *ST_Geometry* is either represented in Big Endian format or Little Endian format. Conversion between Big Endian format or Little Endian format is a simple operation involving reversing the order of octets within each <uint32> or <double> value in the BINARY LARGE OBJECT.

6 Point Types

6.1 ST_Point Type and Routines

6.1.1 ST_Point Type

Purpose

The ST_Point type is a 0-dimensional geometry and represents a single location.

Definition

```
CREATE TYPE ST_Point
   UNDER ST_Geometry
   AS (
      ST_PrivateX DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateY DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateZ DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateM DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Point
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_Point
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Point
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_Point
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Point
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_Point
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Point
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_Point
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    mcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    mcoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_X()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_X
   (xcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Y()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Y
   (ycoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Z()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Z
   (zcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_M()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_M
   (mcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ExplicitPoint()
   RETURNS DOUBLE PRECISION ARRAY[4]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
2) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
3) The attribute *ST_PrivateX* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateX*.
4) The attribute *ST_PrivateY* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateY*.
5) The attribute *ST_PrivateZ* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateZ*.
6) The attribute *ST_PrivateM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateM*.

Description

1) The *ST_Point* type provides for public use:
   a) a method *ST_Point(CHARACTER LARGE OBJECT)*,
   b) a method *ST_Point(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_Point(BINARY LARGE OBJECT)*,
   d) a method *ST_Point(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION)*,
   f) a method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,
   g) a method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)*,
   h) a method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,
   i) a method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)*,
   j) a method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,

k) a method *ST_X()*,
l) a method *ST_X(DOUBLE PRECISION)*,
m) a method *ST_Y()*,
n) a method *ST_Y(DOUBLE PRECISION)*,
o) a method *ST_Z()*,
p) a method *ST_Z(DOUBLE PRECISION)*,
q) a method *ST_M()*,
r) a method *ST_M(DOUBLE PRECISION)*,
s) a method *ST_ExplicitPoint()*,
t) a function *ST_PointFromText(CHARACTER LARGE OBJECT)*,
u) a function *ST_PointFromText(CHARACTER LARGE OBJECT, INTEGER)*,
v) a function *ST_PointFromWKB(BINARY LARGE OBJECT)*,
w) a function *ST_PointFromWKB(BINARY LARGE OBJECT, INTEGER)*,
x) a function *ST_PointFromGML(CHARACTER LARGE OBJECT)*,
y) a function *ST_PointFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateX* attribute contains the x coordinate value.
3) The *ST_PrivateY* attribute contains the y coordinate value.
4) The *ST_PrivateZ* attribute contains the z coordinate value.
5) The *ST_PrivateM* attribute contains the m coordinate value.
6) An *ST_Point* value is a 0-dimensional geometry that represents a single location.
7) The dimension of an *ST_Point* value is 0 (zero).
8) The coordinate dimension of an *ST_Point* value is the number of coordinate values associated with the position.
9) The boundary of an *ST_Point* value is the empty set.
10) An *ST_Point* value is simple.
11) An *ST_Point* value returned by the constructor function corresponds to the empty set.
12) An *ST_Point* value is not well formed if either:
    a) the *ST_PrivateX* attribute is the null value and the *ST_PrivateY* attribute is not the null value,
    b) the *ST_PrivateY* attribute is the null value and the *ST_PrivateX* attribute is not the null value,
    c) the *ST_PrivateIs3D* attribute is 0 (zero) and the *ST_PrivateZ* attribute is not the null value,
    d) the *ST_PrivateIs3D* attribute is 1 (one), the *ST_PrivateX* attribute is the null value, and the *ST_PrivateZ* attribute is not the null value,
    e) the *ST_PrivateIs3D* attribute is 1 (one), the *ST_PrivateX* attribute is not the null value, and the *ST_PrivateZ* attribute is the null value,
    f) the *ST_PrivateIsMeasured* attribute is 0 (zero) and the *ST_PrivateM* attribute is not the null value,
    g) the *ST_PrivateIsMeasured* attribute is 1 (one), the *ST_PrivateX* attribute is the null value, and the *ST_PrivateM* attribute is not the null value,
    h) the *ST_PrivateIsMeasured* attribute is 1 (one), the *ST_PrivateX* attribute is not the null value, and the *ST_PrivateM* attribute is the null value, or
    i) the *ST_PrivateCoordinateDimension* attribute is not equal to value expression: *2 + ST_PrivateIs3D + ST_PrivateIsMeasured*.

6.1.2 ST_Point Methods

Purpose

Return an ST_Point value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified coordinate values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Point
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Point
   FOR ST_Point
   RETURN NEW ST_Point(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_Point
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Point
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Point
   FOR ST_Point
   RETURN NEW ST_Point(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_Point
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Point
   FOR ST_Point
   RETURN ST_PointFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   RETURN NEW ST_Point(xcoord, ycoord, 0)

CREATE CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Point
   FOR ST_Point
   RETURN SELF.                    -- Return an ST_Point value with
      ST_PrivateDimension(0).           -- dimension = 0,
      ST_PrivateCoordinateDimension(2). -- coordinate dimension = 2,
      ST_PrivateIs3D(0).                -- is not 3D,
      ST_PrivateIsMeasured(0).          -- is not measured,
      ST_SRID(ansrid).                   -- SRID = ansrid,
      ST_X(xcoord).                     -- x coordinate = xcoord,
      ST_Y(ycoord)                      -- y coordinate = ycoord
```

```
CREATE CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   RETURN NEW ST_Point(xcoord, ycoord, zcoord, 0)

CREATE CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      IF ( xcoord IS NULL AND ycoord IS NOT NULL ) OR
         ( xcoord IS NOT NULL AND ycoord IS NULL ) THEN
         -- if not well-formed, raise an exception
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      IF xcoord IS NULL THEN
         -- If xcoord is the null value, assume an empty
         -- point value is being created, check zcoord is null.
         IF zcoord IS NOT NULL THEN
            SIGNAL SQLSTATE '2FF16'
               SET MESSAGE_TEXT = 'not an empty set';
         END IF;
      ELSE
         -- Otherwise, check zcoord is not null.
         IF zcoord IS NULL THEN
            SIGNAL SQLSTATE '2FF03'
               SET MESSAGE_TEXT = 'null argument';
         END IF;
      END IF;
      RETURN SELF.                    -- Return an ST_Point value with
         ST_PrivateDimension(0).      -- dimension = 0,
         ST_PrivateCoordinateDimension(
            CASE
               WHEN zcoord IS NOT NULL THEN
            -- if z coordinate is not the null
            -- value, then
                  3                    --   coordinate dimension = 3
               ELSE                    -- otherwise,
                  2                    --   coordinate dimension = 2
            END).
         ST_PrivateIs3D(
            CASE
               WHEN zcoord IS NOT NULL THEN
            -- if z coordinate is not the null
                  1                    --   value, then, is 3D
               ELSE                    -- otherwise,
                  0                    --   is not 3D
            END).
         ST_PrivateIsMeasured(0).      -- not measured,
         ST_SRID(ansrid).               -- SRID = ansrid,
         ST_PrivateX(xcoord).          -- x coordinate = xcoord,
         ST_PrivateY(ycoord).          -- y coordinate = ycoord,
         ST_PrivateZ(zcoord);          -- z coordinate = zcoord
   END
```

```
CREATE CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    mcoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   RETURN NEW ST_Point(xcoord, ycoord, zcoord, mcoord, 0)

CREATE CONSTRUCTOR METHOD ST_Point
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    mcoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      IF ( xcoord IS NULL AND ycoord IS NOT NULL ) OR
         ( xcoord IS NOT NULL AND ycoord IS NULL ) THEN
         -- if not well-formed, raise an exception
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      IF xcoord IS NULL THEN
         -- If xcoord is the null value, assume an empty
         -- point value is being created, check zcoord and mcoord is null.
         IF zcoord IS NOT NULL THEN
            SIGNAL SQLSTATE '2FF16'
               SET MESSAGE_TEXT = 'not an empty set';
         END IF;
      END IF;
      RETURN SELF.                    -- Return an ST_Point value with
         ST_PrivateDimension(0).           -- dimension = 0,
         ST_PrivateCoordinateDimension(
            CASE
               WHEN (zcoord IS NOT NULL AND mcoord IS NOT NULL) THEN
            -- if z coordinate is not the null
            -- value and mcoord is not the null
            -- value, then
                  4                   --   coordinate dimension = 4
               WHEN ((zcoord IS NOT NULL AND mcoord IS NULL) OR
                     (zcoord IS NULL AND mcoord IS NOT NULL)) THEN
            -- if z coordinate is not the null
            -- value and mcoord is the null
            -- value or if z coordinate is
            -- the null value and m is not
            -- the null value, then
                  3                   --   coordinate dimension = 3
               ELSE                   -- otherwise,
                  2                   --   coordinate dimension = 2
            END).
         ST_PrivateIs3D(
            CASE
               WHEN zcoord IS NOT NULL THEN
            -- if z coordinate is not the null
                  1                   --   value, then is 3D
               ELSE                   -- otherwise,
                  0                   --   is not 3D
            END).
         ST_PrivateIsMeasured(
```

```
            CASE
               WHEN mcoord IS NOT NULL THEN
            -- if m coordinate value is
            -- not the null value, then
                  1                    --   is measured
               ELSE                    -- otherwise,
                  0                    --   is not measured
            END).
         ST_SRID(ansrid).               -- SRID = ansrid,
         ST_PrivateX(xcoord).          -- x coordinate = xcoord,
         ST_PrivateY(ycoord).          -- y coordinate = ycoord,
         ST_PrivateZ(zcoord).          -- z coordinate = zcoord
         ST_PrivateM(
            CASE
               WHEN ( xcoord IS NULL AND
                      ycoord IS NULL AND
                      zcoord IS NULL ) THEN
            -- if x, y and z coordinate value
            -- is the null value, then
                  NULL                 --   set m coordinate value to
                  --   the null value
               ELSE                    -- otherwise,
                  mcoord               --   set m coordinate value
                  -- to mcoord
            END);
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
2) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_Point(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_Point(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_Point(awktorgml, 0)*.
3) The method *ST_Point(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_Point(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Point XML element in the GML representation, then return the result of the value expression: *ST_PointFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_PointFromText(awktorgml, ansrid)*.
5) The method *ST_Point(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_Point(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_Point(awkb, 0)*.
7) The method *ST_Point(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a BINARY LARGE OBJECT value *awkb*,

b) an INTEGER value *ansrid*.

8) The n2ull-call type-preserving SQL-invoked constructor method *ST_Point(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_PointFromWKB(awkb, ansrid)*.

9) The method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

a) a DOUBLE PRECISION value *xcoord*,

b) a DOUBLE PRECISION value *ycoord*.

10) The null-call type-preserving SQL-invoked constructor method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Point(xcoord, ycoord, 0)*.

11) The method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

a) a DOUBLE PRECISION value *xcoord*,

b) a DOUBLE PRECISION value *ycoord*,

c) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* returns an *ST_Point* value with:

a) The dimension set to 0 (zero).

b) The coordinate dimension value set to 2.

c) The *ST_PrivateIs3D* attribute set to 0 (zero).

d) The *ST_PrivateIsMeasured* attribute set to 0 (zero).

e) The spatial reference system identifier set to *ansrid*.

f) Using the method *ST_X(DOUBLE PRECISION)*, the x coordinate value is set to *xcoord*.

g) Using the method *ST_Y(DOUBLE PRECISION)*, the y coordinate value is set to *ycoord*.

h) The z coordinate value set to NULL by default clause.

i) The m coordinate value set to NULL by default clause.

13) The method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

a) a DOUBLE PRECISION value *xcoord*,

b) a DOUBLE PRECISION value *ycoord*,

c) a DOUBLE PRECISION value *zcoord*.

14) The null-call type-preserving SQL-invoked constructor method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Point(xcoord, ycoord, zcoord, 0)*.

15) The method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

a) a DOUBLE PRECISION value *xcoord*,

b) a DOUBLE PRECISION value *ycoord*,

c) a DOUBLE PRECISION value *zcoord*,

d) an INTEGER value *ansrid*.

16) For the type-preserving SQL-invoked constructor method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

Case:

a) If *xcoord* is the null value and *ycoord* is not the null value, or if *xcoord* is not the null value and *ycoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If *xcoord* is the null value and *zcoord* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

c) If *xcoord* is not the null value and *zcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

d) Otherwise, return an *ST_Point* value with:

i) The dimension set to 0 (zero).

ii) Case:

1) If *zcoord* is not the null value, then the coordinate dimension value set to 3.

2) Otherwise, the coordinate dimension value set to 2.

iii) Case:

1) If *zcoord* is not the null value, then the *ST_PrivateIs3D* attribute set to 1 (one).

2) Otherwise, the *ST_PrivateIs3D* attribute set to 0 (zero).

iv) The *ST_PrivateIsMeasured* attribute set to 0 (zero).

v) The spatial reference system identifier set to *ansrid*.

vi) The x coordinate value is set to *xcoord*.

vii) The y coordinate value is set to *ycoord*.

viii) The z coordinate value is set to *zcoord*.

ix) The m coordinate value set to NULL by default clause.

17) The method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

a) a DOUBLE PRECISION value *xcoord*,

b) a DOUBLE PRECISION value *ycoord*,

c) a DOUBLE PRECISION value *zcoord*,

d) a DOUBLE PRECISION value *mcoord*.

18) The null-call type-preserving SQL-invoked constructor method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Point(xcoord, ycoord, zcoord, mcoord, 0)*.

19) The method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

a) a DOUBLE PRECISION value *xcoord*,

b) a DOUBLE PRECISION value *ycoord*,

c) a DOUBLE PRECISION value *zcoord*,

d) a DOUBLE PRECISION value *mcoord*,

e) an INTEGER value *ansrid*.

20) For the type-preserving SQL-invoked constructor method *ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

Case:

a) If *xcoord* is the null value and *ycoord* is not the null value, or if *xcoord* is not the null value and *ycoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If *xcoord* is the null value and *zcoord* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

c) Otherwise, return an ST_Point value with:

   i) The dimension set to 0 (zero).

   ii) Case:

      1) If *zcoord* is not the null value and *mcoord* is not the null value, then the coordinate dimension value set to 4.

      2) If *zcoord* is not the null value and *mcoord* is the null value or if *zcoord* is the null value and *mcoord* is not the null value, then the coordinate dimension value set to 3.

      3) Otherwise, the coordinate dimension value set to 2.

   iii) Case:

      1) If *zcoord* is not the null value, then the *ST_PrivateIs3D* attribute set to 1 (one).

      2) Otherwise, the *ST_PrivateIs3D* attribute set to 0 (zero).

   iv) Case:

      1) If *mcoord* is not the null value, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

      2) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

   v) The spatial reference system identifier set to *ansrid*.

   vi) The x coordinate value is set to *xcoord*.

   vii) The y coordinate value is set to *ycoord*.

   viii) The z coordinate value is set to *zcoord*.

   ix) Case:

      1) If *xcoord* is the null value and *ycoord* is the null value and *zcoord* is the null value, then the m coordinate value is set to the null value.

      2) Otherwise, the m coordinate value is set to *mcoord*.

6.1.3 ST_X Methods

Purpose

Observe and mutate the x coordinate value of an ST_Point value.

Definition

```
CREATE METHOD ST_X()
   RETURNS DOUBLE PRECISION
   FOR ST_Point
   RETURN SELF.ST_PrivateX

CREATE METHOD ST_X
   (xcoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      IF xcoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateX(xcoord)
            END;
      END IF;
   END
```

Description

1) The method *ST_X()* has no input parameters.
2) The null-call method *ST_X()* returns the value of the *ST_PrivateX* attribute.
3) The method *ST_X(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *xcoord*.
4) For the type-preserving method *ST_X(DOUBLE PRECISION)*:

   Case:

   a) If *xcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateX(xcoord)*.

6.1.4 ST_Y Methods

Purpose

Observe and mutate the y coordinate value of an ST_Point value.

Definition

```
CREATE METHOD ST_Y()
   RETURNS DOUBLE PRECISION
   FOR ST_Point
   RETURN SELF.ST_PrivateY

CREATE METHOD ST_Y
   (ycoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      IF ycoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateY(ycoord)
            END;
      END IF;
   END
```

Description

1) The method *ST_Y()* has no input parameters.
2) The null-call method *ST_Y()* returns the value of the *ST_PrivateY* attribute.
3) The method *ST_Y(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *ycoord*.
4) For the type-preserving method *ST_Y(DOUBLE PRECISION)*:

   Case:

   a) If *ycoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateY(ycoord)*.

6.1.5 ST_Z Methods

Purpose

Observe and mutate the z coordinate value of an ST_Point value.

Definition

```
CREATE METHOD ST_Z()
   RETURNS DOUBLE PRECISION
   FOR ST_Point
   RETURN SELF.ST_PrivateZ

CREATE METHOD ST_Z
   (zcoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      IF SELF.ST_IsEmpty() = 0 AND zcoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      IF SELF.ST_IsEmpty() = 1 AND zcoord IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN
         CASE
            WHEN SELF IS NULL THEN NULL
            ELSE SELF.ST_PrivateZ(zcoord)
         END;
   END
```

Description

1) The method *ST_Z()* has no input parameters.
2) The null-call method *ST_Z()* returns the value of the *ST_PrivateZ* attribute.
3) The method *ST_Z(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *zcoord*.
4) For the type-preserving method *ST_Z(DOUBLE PRECISION)*:

   Case:

   a) If SELF is an empty set and *zcoord* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument.*
   b) If SELF is not an empty set and *zcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set.*
   c) If SELF is the null value, then return the null value.
   d) Otherwise, return the result of the value expression: *SELF.ST_PrivateZ(zcoord)*.

6.1.6 ST_M Methods

Purpose

Observe and mutate the m coordinate value of an ST_Point value.

Definition

```
CREATE METHOD ST_M()
   RETURNS DOUBLE PRECISION
   FOR ST_Point
   RETURN SELF.ST_PrivateM

CREATE METHOD ST_M
   (mcoord DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Point
   BEGIN
      IF SELF.ST_IsEmpty() = 0 AND mcoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      IF SELF.ST_IsEmpty() = 1 AND mcoord IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      RETURN
         CASE
            WHEN SELF IS NULL THEN NULL
            ELSE SELF.ST_PrivateM(mcoord)
         END;
   END
```

Description

1) The method *ST_M()* has no input parameters.

2) The null-call method *ST_M()* returns the value of the *ST_PrivateM* attribute.

3) The method *ST_M(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *mcoord*.

4) For the type-preserving method *ST_M(DOUBLE PRECISION)*:

   Case:

   a) If SELF is an empty set and *mcoord* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument.*

   b) If SELF is not an empty set and *mcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set.*

   c) If SELF is the null value, then return the null value.

   d) Otherwise, return the result of the value expression: *SELF.ST_PrivateM(mcoord)*.

6.1.7 ST_ExplicitPoint Method

Purpose

Return the coordinate values as a DOUBLE PRECISION ARRAY value.

Definition

```
CREATE METHOD ST_ExplicitPoint()
   RETURNS DOUBLE PRECISION ARRAY[4]
   FOR ST_Point
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         WHEN (SELF.ST_Z() IS NOT NULL AND
               SELF.ST_M() IS NOT NULL) THEN
            ARRAY[SELF.ST_X(), SELF.ST_Y(), SELF.ST_Z(), SELF.ST_M()]
         WHEN (SELF.ST_Z() IS NOT NULL AND
               SELF.ST_M() IS NULL) THEN
            ARRAY[SELF.ST_X(), SELF.ST_Y(), SELF.ST_Z()]
         WHEN (SELF.ST_Z() IS NULL AND
               SELF.ST_M() IS NOT NULL) THEN
            ARRAY[SELF.ST_X(), SELF.ST_Y(), SELF.ST_M()]
         ELSE
            ARRAY[SELF.ST_X(), SELF.ST_Y()]
      END
```

Description

1) The method *ST_ExplicitPoint()* has no input parameters.

2) For the null-call method *ST_ExplicitPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If the z coordinate value is not the null value and the m coordinate value is not the null value, then return an array of type DOUBLE PRECISION with the first element representing the x coordinate value, the second element representing the y coordinate value, the third element representing the z coordinate value, and the forth element representing the m coordinate value.

   c) If the z coordinate value is not the null value and the m coordinate value is the null value, then return an array of type DOUBLE PRECISION with the first element representing the x coordinate value, the second element representing the y coordinate value, and the third element representing the z coordinate value.

   d) If the z coordinate value is the null value and the m coordinate value is not the null value, then return an array of type DOUBLE PRECISION with the first element representing the x coordinate value, the second element representing the y coordinate value, and the third element representing the m coordinate value.

   e) Otherwise, return an array of type DOUBLE PRECISION with the first element representing the x coordinate value and the second element representing the y coordinate value.

6.1.8 ST_PointFromText Functions

Purpose

Return an ST_Point value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Point value.

Definition

```
CREATE FUNCTION ST_PointFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PointFromText(awkt, 0)

CREATE FUNCTION ST_PointFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_PointFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_PointFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_PointFromText(awkt, 0)*.
3) The function *ST_PointFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_PointFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Point* value.

      If *awkt* is not producible in the BNF for <point text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_Point)*.

6.1.9 ST_PointFromWKB Functions

Purpose

Return an ST_Point value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Point value.

Definition

```
CREATE FUNCTION ST_PointFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PointFromWKB(awkb, 0)

CREATE FUNCTION ST_PointFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_PointFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_PointFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_PointFromWKB(awkb, 0)*.
3) The function *ST_PointFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_PointFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_Point* value.

      If *awkb* is not producible in the BNF for <point binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_Point)*.

6.1.10 ST_PointFromGML Functions

Purpose

Return an ST_Point value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Point.

Definition

```
CREATE FUNCTION ST_PointFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PointFromGML(agml, 0)

CREATE FUNCTION ST_PointFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_PointFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_PointFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_PointFromGML(agml, 0)*.
3) The function *ST_PointFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_PointFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a Point XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_Point)*.

7 Curve Types

7.1 ST_Curve Type and Routines

7.1.1 ST_Curve Type

Purpose

The ST_Curve type is a supertype for 1-dimensional geometry types and represents a continuous locus of points from the start point to the end point. Subtypes of ST_Curve specify the form of interpolation between points.

Definition

```
CREATE TYPE ST_Curve
   UNDER ST_Geometry
   NOT INSTANTIABLE
   NOT FINAL

   METHOD ST_Length()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Length
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DLength()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DLength
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_StartPoint()
      RETURNS ST_Point
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_EndPoint()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsClosed()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DIsClosed()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsRing()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DIsRing()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_CurveToLine()
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DistanceToPoint
   (apoint ST_Point)
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DistanceToPoint
   (apoint ST_Point,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_3DDistanceToPt(apoint ST_Point)
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DDistanceToPt
   (apoint ST_Point,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointAtDistance
   (adistance DOUBLE PRECISION)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointAtDistance
   (adistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DPtAtDistance
   (adistance DOUBLE PRECISION)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DPtAtDistance
   (adistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PerpPoints
   (apoint ST_Point)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The *ST_Curve* type provides for public use:
   a) a method *ST_Length()*,
   b) a method *ST_Length(CHARACTER VARYING)*,
   c) a method *ST_3DLength()*,
   d) a method *ST_3DLength(CHARACTER VARYING)*,
   e) a method *ST_StartPoint()*,
   f) a method *ST_EndPoint()*,
   g) a method *ST_IsClosed()*,
   h) a method *ST_3DIsClosed()*,
   i) a method *ST_IsRing()*,
   j) a method *ST_3DIsRing()*,
   k) a method *ST_CurveToLine()*,
   l) a method *ST_DistanceToPoint(ST_Point)*,
   m) a method *ST_DistanceToPoint(ST_Point, CHARACTER VARYING)*,
   n) a method *ST_3DDistanceToPt(ST_Point)*,
   o) a method *ST_3DDistanceToPt(ST_Point, CHARACTER VARYING)*,
   p) a method *ST_PointAtDistance(DOUBLE PRECISION)*,
   q) a method *ST_PointAtDistance(DOUBLE PRECISION, CHARACTER VARYING)*,
   r) a method *ST_3DPtAtDistance(DOUBLE PRECISION)*,
   s) a method *ST_3DPtAtDistance(DOUBLE PRECISION, CHARACTER VARYING)*,
   t) a method *ST_PerpPoints(ST_Point)*.
2) An *ST_Curve* value is a 1-dimensional geometry that represents a continuous, connected locus of points from the start point to the end point.
3) Subtypes of the *ST_Curve* type specifies the form of interpolation between *ST_Point* values.
4) An *ST_Curve* value is defined to be topologically closed.
5) An *ST_Curve* value is the homomorphic image of a real, closed interval:

   Domain = [a, b] = $\{ x \in \mathrm{R} \mid a <= x <= b \}$ under a mapping f:[a, b] $\rightarrow \mathrm{R}^2$.
6) An *ST_Curve* value is not simple if any interior point has the same location as another interior point or a point in the boundary:

   $\forall$ c $\in$ *ST_Curve*, [a, b] = c.Domain,

   c.*ST_IsSimple()* $\Leftrightarrow$
   $(\forall x_1, x_2 \in (a, b]\ x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)) \wedge (\forall x_1, x_2 \in [a, b)\ x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2))$
   a) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation of *ST_IsSimple*.
   b) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation of *ST_3DIsSimple*.
   c) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.
7) The dimension of an *ST_Curve* value is 1 (one).
8) The start point of an *ST_Curve* value is returned by the method *ST_StartPoint()*.
9) The end point of an *ST_Curve* value is returned by the method *ST_EndPoint()*.

10) If the start point of an *ST_Curve* value is equal to the end point of the *ST_Curve* value, then the *ST_Curve* value is closed.
    a) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation of *ST_IsClosed*.
    b) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation of *ST_3DIsClosed*.
    c) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.
11) The boundary of a closed *ST_Curve* value is the empty set.
12) The boundary of an *ST_Curve* value that is not closed consists of the start point and end point of the *ST_Curve* value.
13) If an *ST_Curve* value is simple and closed, then it is called a *ring*.
    a) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation of *ST_IsRing*.
    b) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation of *ST_3DIsRing*.
    c) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

7.1.2 ST_Length Methods

Purpose

Return the length measurement of an ST_Curve value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Length()
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_Length
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Length()* has no input parameters.
2) For the null-call method *ST_Length()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined length of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Length()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.
3) The method *ST_Length(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_Length(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) Otherwise, return the implementation-defined length of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

7.1.3 ST_3DLength Methods

Purpose

Return the length measurement of an ST_Curve value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DLength()
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE METHOD ST_3DLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DLength()* has no input parameters.
2) For the null-call method *ST_3DLength()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined length of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DLength()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.
3) The method *ST_3DLength(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_3DLength(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) Otherwise, return the implementation-defined length of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

7.1.4 ST_StartPoint Method

Purpose

Return an ST_Point value that is the start point of an ST_Curve value including existing z and m coordinate values in the resultant geometry.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_StartPoint()* has no input parameters.
2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return an *ST_Point* value that is the start point of SELF.
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate value is included in the resultant geometry.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate value is included in the resultant geometry.

7.1.5 ST_EndPoint Method

Purpose

Return an ST_Point value that is the end point of an ST_Curve value including existing z and m coordinate values in the resultant geometry.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_EndPoint()* has no input parameters.
2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_Point* value that is the end point of SELF.
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate value is included in the resultant geometry.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate value is included in the resultant geometry.

7.1.6 ST_IsClosed Method

Purpose

Test if an ST_Curve value is closed, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_IsClosed()
   RETURNS INTEGER
   FOR ST_Curve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            0
         ELSE
            SELF.ST_Boundary().ST_IsEmpty()
      END
```

Description

1) The method *ST_IsClosed()* has no input parameters.
2) For the null-call method *ST_IsClosed()*:

   Case:

   a) If SELF is an empty set, then return 0 (zero).
   b) If the boundary of the *ST_Curve* value is the empty set, then 1 (one).
   c) Otherwise, 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

7.1.7 ST_3DIsClosed Method

Purpose

Test if an ST_Curve value is closed, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DIsClosed()
   RETURNS INTEGER
   FOR ST_Curve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            0
         ELSE
            SELF.ST_3DBoundary().ST_IsEmpty()
      END
```

Description

1) The method *ST_3DIsClosed()* has no input parameters.
2) For the null-call method *ST_3DIsClosed()*:

   Case:

   a) If SELF is an empty set, then return 0 (zero).
   b) If the boundary of the *ST_Curve* value is the empty set, then 1 (one).
   c) Otherwise, 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

7.1.8 ST_IsRing Method

Purpose

Test if an ST_Curve value is a ring, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_IsRing()
   RETURNS INTEGER
   FOR ST_Curve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            0
         WHEN (SELF.ST_IsSimple() = 1 AND SELF.ST_IsClosed() = 1) THEN
            1
         ELSE
            0
      END
```

Description

1) The method *ST_IsRing()* has no input parameters.
2) For the null-call method *ST_IsRing()*:

   Case:

   a) If SELF is an empty set, then return 0 (zero).
   b) If SELF is simple and SELF is closed, then return 1 (one).
   c) Otherwise 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

7.1.9 ST_3DIsRing Method

Purpose

Test if an ST_Curve value is a ring, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DIsRing()
   RETURNS INTEGER
   FOR ST_Curve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            0
         WHEN (SELF.ST_3DIsSimple() = 1 AND SELF.ST_3DIsClosed() = 1) THEN
            1
         ELSE
            0
      END
```

Description

1) The method *ST_3DIsRing()* has no input parameters.
2) For the null-call method *ST_3DIsRing()*:

   Case:

   a) If SELF is an empty set, then return 0 (zero).
   b) If SELF is simple and SELF is closed, then return 1 (one).
   c) Otherwise 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

7.1.10 ST_CurveToLine Method

Purpose

Return the ST_LineString value approximation of an ST_Curve value, considering z and m coordinate values in the calculations and including z and m coordinate values in the resultant geometry.

Definition

```
CREATE METHOD ST_CurveToLine()
   RETURNS ST_LineString
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_CurveToLine()* has no input parameters.
2) For the null-call method *ST_CurveToLine()*:

   Case:

   a) If SELF is an empty set, then return an empty set of type *ST_LineString*.
   b) Otherwise, return the implementation-defined *ST_LineString* value approximation of the *ST_Curve* value.
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation and are included in the resultant geometry.
4) If *SELF.ST_IsMeasured ()* is equal to 1 (one), then m coordinate values are calculated for the *ST_LineString.ST_PrivatePoints ST_Point* values by linear interpolation based on curve length using an implementation-defined interpolation algorithm. The resultant m coordinate values are included in the resultant geometry.

7.1.11 ST_DistanceToPoint Methods

Purpose

Return the distance from the start of the curve measured along the curve to a point on the curve, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_DistanceToPoint
   (apoint ST_Point)
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_DistanceToPoint
   (apoint ST_Point,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_DistanceToPoint(ST_Point)* takes the following input parameter:
   a) an *ST_Point* value *apoint*.
2) For the null-call method *ST_DistanceToPoint(ST_Point)*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *apoint* is an empty set, then return the null value.
      iii) If SELF and *apoint* do not spatially intersect such that z and m coordinate values are not considered in the calculation, then an exception condition is raised: *SQL/MM Spatial exception – the point is not on the curve*.
      iv) Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_DistanceToPoint(ST_Point)* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_DistanceToPoint(ST_Point)* is in an implementation-defined unit of measure.
3) The method *ST_DistanceToPoint(ST_Point, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_Point* value *apoint*.

b) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_DistanceToPoint(ST_Point, CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the distance along the curve SELF from the start point to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) If *apoint* is an empty set, then return the null value.

iii) If SELF and *apoint* do not spatially intersect such that z and m coordinate values are not considered in the calculation, then an exception condition is raised: *SQL/MM Spatial exception – the point is not on the curve*.

iv) Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.

e) The returned value is measured in the units indicated by *aunit*.

7.1.12 ST_3DDistanceToPt Methods

Purpose

Return the distance from the start of the curve measured along the curve to a point on the curve, considering z (but not m) coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DDistanceToPt
   (apoint ST_Point)
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE METHOD ST_3DDistanceToPt
   (apoint ST_Point,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DDistanceToPt(ST_Point)* takes the following input parameter:
   a) an *ST_Point* value *apoint*.
2) For the null-call method *ST_3DDistanceToPt(ST_Point)*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *apoint* is an empty set, then return the null value.
      iii) If SELF and *apoint* do not spatially intersect such that z (but not m) coordinate values are considered in the calculation, then an exception condition is raised: *SQL/MM Spatial exception – the point is not on the curve*.
      iv) Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are considered in the calculation.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DDistanceToPt(ST_Point)* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_3DDistanceToPt(ST_Point)* is in an implementation-defined unit of measure.
3) The method *ST_3DDistanceToPt(ST_Point, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_Point* value *apoint*.

b) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_3DDistanceToPt(ST_Point, CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the distance along the curve SELF from the start point to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) If *apoint* is an empty set, then return the null value.

iii) If *SELF.ST_Is3D()* is equal to 0 (zero) or *apoint.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial Exception – both geometries must be 3D*.

iv) If SELF and *apoint* do not spatially intersect such that z (but not m) coordinate values are considered in the calculation, then an exception condition is raised: *SQL/MM Spatial exception – the point is not on the curve*.

v) Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are considered in the calculation.

e) The returned value is measured in the units indicated by *aunit*.

7.1.13 ST_PointAtDistance Methods

Purpose

Return the ST_Point value that is the specified distance from the start of the curve measured along the curve, ignoring z coordinate values in the calculations and including a z and interpolated m coordinate in the return value.

Definition

```
CREATE METHOD ST_PointAtDistance
   (adistance DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_PointAtDistance
   (adistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Point
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_PointAtDistance(DOUBLE PRECISION)* takes the following input parameter:
   a) a DOUBLE PRECISION value *adistance*.
2) For the null-call method *ST_PointAtDistance(DOUBLE PRECISION)*:
   a) The parameter *adistance* is measured in an implementation-defined linear unit of measure in the spatial reference system of SELF.
   b) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *SELF.ST_Length()* is less than *adistance*, then an exception condition is raised: *SQL/MM Spatial exception – the given distance is longer than curve*.
      iii) Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are considered in the calculations and a z and interpolated m coordinate is included in the return value.
3) The method *ST_PointAtDistance(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:
   a) a DOUBLE PRECISION value *adistance*.
   b) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_PointAtDistance(DOUBLE PRECISION, CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to measure the *adistance* along the curve SELF from the start point, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) If *SELF.ST_Length(aunit)* is less than *adistance*, then an exception condition is raised: *SQL/MM Spatial exception – the given distance is longer than curve*.

   iii) Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are considered in the calculations and a z and interpolated m coordinate is included in the return value.

7.1.14 ST_3DPtAtDistance Methods

Purpose

Return the ST_Point value that is the specified distance from the start of the curve measured along the curve, considering z coordinate values in the calculations and including a z and interpolated m coordinate in the return value.

Definition

```
CREATE METHOD ST_3DPtAtDistance
   (adistance DOUBLE PRECISION)
   RETURNS ST_Point
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE METHOD ST_3DPtAtDistance
   (adistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Point
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DPtAtDistance(DOUBLE PRECISION)* takes the following input parameter:
   a) a DOUBLE PRECISION value *adistance*.
2) For the null-call method *ST_3DPtAtDistance(DOUBLE PRECISION)*:
   a) The parameter *adistance* is measured in an implementation-defined linear unit of measure in the spatial reference system of SELF.
   b) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *SELF.ST_3DLength()* is less than *adistance*, then an exception condition is raised: *SQL/MM Spatial exception – the given distance is longer than curve*.
      iii) Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are considered in the calculations and a z and interpolated m coordinate is included in the return value.
3) The method *ST_3DPtAtDistance(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:
   a) a DOUBLE PRECISION value *adistance*.
   b) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_3DPtAtDistance(DOUBLE PRECISION, CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to measure the *adistance* along the curve SELF from the start point, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) If *SELF.ST_3DLength(aunit)* is less than *adistance*, then an exception condition is raised: *SQL/MM Spatial exception – the given distance is longer than curve*.

   iii) Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are considered in the calculations and a z and interpolated m coordinate is included in the return value.

7.1.15 ST_PerpPoints Method

Purpose

Return the geometry representing the perpendicular projection of the given point onto the curve, ignoring z coordinate values in the calculations and including interpolated m (but not z) coordinates in the resultant geometry.

Definition

```
CREATE METHOD ST_PerpPoints
   (apoint ST_Point)
   RETURNS ST_Geometry
   FOR ST_Curve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_PerpPoints(ST_Point)* takes the following input parameter:

   a) an *ST_Point* value *apoint*.

2) For the null-call method *ST_PerpPoints(ST_Point)*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If *apoint* is an empty set, then return the null value.

   c) If SELF and *apoint* spatially intersect such that z and m coordinate values are not considered in the calculation, then return *apoint*.

   d) If *apoint* cannot be perpendicularly projected on SELF, then return an empty set.

   e) Otherwise, return a geometry value representing the perpendicular projection of *apoint* on SELF, calculated in the spatial reference system of SELF, using an implementation-defined algorithm such that z coordinate values are not considered in the calculation and interpolated m (but not z) coordinates are included in the return values.

      NOTE The result of the projection algorithm may produce the following
      - an ST_Point value when it produces a single point result
      - an ST_MultiPoint value when it produces a finite number of points
      - an ST_Curve value when it produces a connected set of points
      - an ST_MultiCurve value when it produces a number of connected set of points
      - an ST_GeomCollection when it produces a mixture of point values and curve values.

7.2 ST_LineString Type and Routines

7.2.1 ST_LineString Type

Purpose

The ST_LineString type is a subtype of the ST_Curve type and represents a continuous locus of points from the start point to the end point with a linear interpolation between points.

Definition

```
CREATE TYPE ST_LineString
   UNDER ST_Curve
   AS (
      ST_PrivatePoints ST_Point
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LineString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_LineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LineString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_LineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LineString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_LineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LineString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_LineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```



```
CONSTRUCTOR METHOD ST_LineString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_LineString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_LineString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_LineString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_LineString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumPoints()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivatePoints* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePoints*.

Description

1) The *ST_LineString* type provides for public use:
   a) a method *ST_LineString(CHARACTER LARGE OBJECT)*,
   b) a method *ST_LineString(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_LineString(BINARY LARGE OBJECT)*,
   d) a method *ST_LineString(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_LineString(ST_Point ARRAY)*,
   f) a method *ST_LineString(ST_Point ARRAY, INTEGER)*,
   g) a method *ST_Points()*,
   h) a method *ST_Points(ST_Point ARRAY)*,
   i) a method *ST_NumPoints()*,
   j) a method *ST_PointN(INTEGER)*,
   k) an overriding method *ST_StartPoint()*,
   l) an overriding method *ST_EndPoint()*,
   m) a function *ST_LineFromText(CHARACTER LARGE OBJECT)*,
   n) a function *ST_LineFromText(CHARACTER LARGE OBJECT, INTEGER)*,
   o) a function *ST_LineFromWKB(BINARY LARGE OBJECT)*,
   p) a function *ST_LineFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   q) a function *ST_LineFromGML(CHARACTER LARGE OBJECT)*,
   r) a function *ST_LineFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The *ST_PrivatePoints* attribute contains the collection of *ST_Point* values.
3) The *ST_PrivatePoints* attribute shall not be the null value. The elements in the *ST_PrivatePoints* attribute shall not be the null value.
4) If the cardinality of the *ST_PrivatePoints* attribute is greater than or equal to two, then the *ST_LineString* value is well formed.
5) All the *ST_Point* values in the *ST_PrivatePoints* attribute shall be in the same spatial reference system as the *ST_LineString* value.
6) The coordinate dimension of an *ST_LineString* value is equal to the coordinate dimension of its *ST_Point* values.
7) The type *ST_LineString* is a subtype of *ST_Curve* with linear interpolation between points. Each consecutive pair of points defines a *line segment*.
8) If the cardinality of the *ST_PrivatePoints* attribute is two, then the *ST_LineString* value is called a *line*.
9) If an *ST_LineString* value is simple and closed, then it is called a *linear ring*.
10) An *ST_LineString* value returned by the constructor function corresponds to the empty set.
11) An *ST_LineString* value with the cardinality of the *ST_PrivatePoints* attribute equal to 0 (zero) corresponds to the empty set.

7.2.2 ST_LineString Methods

Purpose

Return an ST_LineString value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Point values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LineString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_LineString
   FOR ST_LineString
   RETURN NEW ST_LineString(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_LineString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_LineString
   FOR ST_LineString
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LineString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_LineString
   FOR ST_LineString
   RETURN NEW ST_LineString(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_LineString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_LineString
   FOR ST_LineString
   RETURN ST_LineFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_LineString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_LineString
   FOR ST_LineString
   RETURN SELF.ST_SRID(0).ST_Points(apointarray)

CREATE CONSTRUCTOR METHOD ST_LineString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_LineString
   FOR ST_LineString
   RETURN SELF.ST_SRID(ansrid).ST_Points(apointarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_LineString(CHARACTER LARGE OBJECT)* takes the following input parameter:

a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_LineString(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_LineString(awktorgml, 0)*.

3) The method *ST_LineString(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_LineString(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a LineString or LineStringSegment XML element in the GML representation, then return the result of the value expression: *ST_LineFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_LineFromText(awktorgml, ansrid)*.

5) The method *ST_LineString(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_LineString(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_LineString(awkb, 0)*.

7) The method *ST_LineString(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_LineString(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_LineFromWKB(awktorgml, ansrid)*.

9) The method *ST_LineString(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_LineString(ST_Point ARRAY)* returns an *ST_LineString* value with:

   a) The spatial reference system identifier set to 0 (zero).

   b) Using the method *ST_Points(ST_Point ARRAY):*

      i) the *ST_PrivateDimension* attribute set to 1 (one).

      ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

      iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

      iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

      v) the *ST_PrivatePoints* attribute set to *apointarray*.

11) The method *ST_LineString(ST_Point ARRAY, INTEGER)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*,

   b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_LineString(ST_Point ARRAY, INTEGER)* returns an *ST_LineString* value with:

   a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Points(ST_Point ARRAY)*:

   i) the *ST_PrivateDimension* attribute set to 1 (one).

   ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

   iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

   iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

   v) the *ST_PrivatePoints* attribute set to *apointarray*.

7.2.3 ST_Points Methods

Purpose

Observe and mutate the ST_PrivatePoints attribute of an ST_LineString value.

Definition

```
CREATE METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_LineString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivatePoints
      END

CREATE METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_LineString
   FOR ST_LineString
   BEGIN
      -- If apointarray is the null value, contains null elements, or
      -- contains consecutive duplicate points, then raise an exception.
      CALL ST_CheckConsecDups(apointarray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_LineString);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and apointarray.
      IF (CARDINALITY(apointarray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(apointarray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      RETURN
         SELF.ST_PrivateDimension(1).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(apointarray)).
            ST_PrivateIs3D(ST_GetIs3D(apointarray)).
            ST_PrivateIsMeasured(ST_GetIsMeasured(apointarray)).
            ST_PrivatePoints(apointarray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Points()* has no input parameters.
2) For the null-call method *ST_Points()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivatePoints* attribute of SELF.
3) The method *ST_Points(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*.

4) For the type-preserving method *ST_Points(ST_Point ARRAY)*:

   a) Call the procedure *ST_CheckConsecDups(ST_Geometry ARRAY)* to check if *apointarray* is the null value, contains null elements, or contains consecutive duplicate points.

   b) Case:

      i) If SELF is the null value, then return the null value.

      ii) If the cardinality of *apointarray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID(apointarray)*, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

      iii) Otherwise, return an *ST_LineString* value with:

         1) The dimension set to 1 (one).

         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(apointarray)*.

         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

         5) The *ST_PrivatePoints* attribute set to *apointarray*.

7.2.4 ST_NumPoints Method

Purpose

Return the cardinality of the ST_PrivatePoints attribute of an ST_LineString value.

Definition

```
CREATE METHOD ST_NumPoints()
   RETURNS INTEGER
   FOR ST_LineString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivatePoints)
      END
```

Description

1) The method *ST_NumPoints()* has no input parameters.
2) For the null-call method *ST_NumPoints()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the cardinality of the *ST_PrivatePoints* attribute.

7.2.5 ST_PointN Method

Purpose

Return the specified element in the ST_PrivatePoints attribute of an ST_LineString value.

Definition

```
CREATE METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   FOR ST_LineString
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Point);
      END IF;
      IF aposition < 1 OR
         aposition > SELF.ST_NumPoints() THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Point);
         END;
      END IF;
      RETURN SELF.ST_PrivatePoints[aposition];
   END
```

Description

1) The method *ST_PointN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_PointN(INTEGER)*:

   Case:
   a) If SELF is an empty set, then return the null value.
   b) If *aposition* is less than 1 (one) or greater than the cardinality of the *ST_PrivatePoints* attribute, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return an *ST_Point* value at element *aposition* in the *ST_PrivatePoints* attribute of SELF.

7.2.6 ST_StartPoint Method

Purpose

Return the start point of an ST_LineString value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_LineString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[1]
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.
2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[1]*.

7.2.7 ST_EndPoint Method

Purpose

Return the end point of an ST_LineString value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_LineString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[SELF.ST_NumPoints()]
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.
2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[SELF.ST_NumPoints()]*.

7.2.8 ST_LineFromText Functions

Purpose

Return an ST_LineString value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LineString value.

Definition

```
CREATE FUNCTION ST_LineFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_LineFromText(awkt, 0)

CREATE FUNCTION ST_LineFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_LineFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_LineFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_LineFromText*(*awkt*, 0).
3) The function *ST_LineFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_LineFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_LineString* value.

      If *awkt* is not producible in the BNF for <linestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_LineString)*.

7.2.9 ST_LineFromWKB Functions

Purpose

Return an ST_LineString value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_LineString value.

Definition

```
CREATE FUNCTION ST_LineFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_LineFromWKB(awkb,0)

CREATE FUNCTION ST_LineFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_LineFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_LineFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_LineFromWKB(awkb, 0)*.
3) The function *ST_LineFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_LineFromWKB(BINARY LARGE OBJECT, INTEGER)*:
   a) The parameter *awkb* is the well-known binary representation of an *ST_LineString* value.

      If *awkb* is not producible in the BNF for <linestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_LineString)*.

7.2.10 ST_LineFromGML Functions

Purpose

Return an ST_LineString value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML LineString or LineStringSegment representation of an ST_LineString value.

Definition

```
CREATE FUNCTION ST_LineFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_LineFromGML(agml, 0)
```

```
CREATE FUNCTION ST_LineFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_LineFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_LineFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression *ST_LineFromGML(agml, 0)*.
3) The function *ST_LineFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_LineFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a LineString or LineStringSegment XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_LineString)*.

7.3 ST_CircularString Type and Routines

7.3.1 ST_CircularString Type

Purpose

The ST_CircularString type is a subtype of the ST_Curve type and represents a continuous locus of points from the start point to the end point with a circular interpolation between points.

Definition

```
CREATE TYPE ST_CircularString
   UNDER ST_Curve
   AS (
      ST_PrivatePoints ST_Point
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_CircularString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_CircularString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CircularString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_CircularString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CircularString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_CircularString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CircularString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_CircularString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    abulgearray DOUBLE PRECISION
     ARRAY[ST_MaxDoublePrecisionArrayElements],
    anormalarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    abulgearray DOUBLE PRECISION
     ARRAY[ST_MaxDoublePrecisionArrayElements],
    anormalarray ST_Vector ARRAY[ST_MaxVectorArrayElements],
    ansrid INTEGER)
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CircularString
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle)
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_CircularString
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    ansrid INTEGER)
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CircularString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumPoints()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_NumSegments()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SegmentN
   (aposition INTEGER)
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_MidPointRep()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Bulge()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_BulgeNormal()
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Center()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Radius()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Radius
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartAngle()
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndAngle()
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,
```

```
OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxDoublePrecisionArrayElements* is the implementation-defined maximum cardinality of an array of DOUBLE PRECISION values.
3) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.
4) ST_MaxUnitNameLength is the implementation-defined maximum length used for the character representation of a unit indication.
5) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
6) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
7) The attribute *ST_PrivatePoints* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePoints*.

Description

1) The *ST_CircularString* type provides for public use:
   a) a method *ST_CircularString(CHARACTER LARGE OBJECT)*,
   b) a method *ST_CircularString(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_CircularString(BINARY LARGE OBJECT)*,
   d) a method *ST_CircularString(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_CircularString(ST_Point ARRAY)*,
   f) a method *ST_CircularString(ST_Point ARRAY, INTEGER)*,
   g) a method *ST_CircularString(ST_Point ARRAY, DOUBLE PRECISION ARRAY, ST_Vector ARRAY)*,
   h) a method *ST_CircularString(ST_Point ARRAY, DOUBLE PRECISION ARRAY, ST_Vector ARRAY, INTEGER)*,
   i) a method *ST_CircularString(ST_Point, DOUBLE PRECISION, ST_Angle, ST_Angle)*,
   j) a method *ST_CircularString(ST_Point, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER)*,
   k) a method *ST_Points()*,
   l) a method *ST_Points(ST_Point ARRAY)*,
   m) a method *ST_NumPoints()*,
   n) a method *ST_PointN(INTEGER)*,
   o) a method *ST_NumSegments()*,
   p) a method *ST_SegmentN(INTEGER)*,
   q) a method *ST_MidPointRep()*,
   r) a method *ST_Bulge()*,
   s) a method *ST_BulgeNormal()*,
   t) a method *ST_Center()*,
   u) a method *ST_Radius()*,
   v) a method *ST_Radius(CHARACTER VARYING)*,

w) a method *ST_StartAngle()*,

x) a method *ST_EndAngle()*,

y) an overriding method *ST_StartPoint()*,

z) an overriding method *ST_EndPoint()*,

aa) a function *ST_CircularFromTxt(CHARACTER LARGE OBJECT)*,

ab) a function *ST_CircularFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,

ac) a function *ST_CircularFromWKB(BINARY LARGE OBJECT)*,

ad) a function *ST_CircularFromWKB(BINARY LARGE OBJECT, INTEGER)*,

ae) a function *ST_CircularFromGML(CHARACTER LARGE OBJECT)*,

af) a function *ST_CircularFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivatePoints* attribute contains the collection of *ST_Point* values.

3) The *ST_PrivatePoints* attribute shall not be the null value. The elements in the *ST_PrivatePoints* attribute shall not be the null value.

4) All the *ST_Point* values in the *ST_PrivatePoints* attribute shall be in the same spatial reference system as the *ST_CircularString* value.

5) The coordinate dimension of an *ST_CircularString* value is equal to the coordinate dimension of its *ST_Point* values.

6) An *ST_CircularString* value consists of one or more circular arc segments connected end to end. The first segment is defined by three points. The first point is the start point of the arc segment. The second point is any intermediate point on the arc segment other than the start or end point. The third point is the end point of the arc segment and shall be distinct from the first point. Subsequent segments are defined by their intermediate and end points only, as the start point is implicitly defined as the previous segment's end point. The distinctness constraint for the start and end points also applies to these subsequent segments.

7) Let *NSEG* be the number of circular arc segments in the *ST_CircularString* value. If *SELF.NumPoints* is equal to 2 * *NSEG* + 1, then the *ST_CircularString* value is well formed.

8) A circular arc segment is the locus of points defined as follows:

   Case:

   a) If the start, intermediate, and end points of an arc segment are not collinear, then the circular arc segment is the locus of points a distance *R* from the center of the arc, beginning at the start point, passing through the intermediate point, and ending at the end point of the circular arc segment. The distance *R* is the radius of the circular arc segment, and is equal to the distance from the center of the circular arc segment and the start, intermediate, or end points. The center of the circular arc segment is defined as follows:

      i) Let *CHORD1* be the line connecting the start point of a circular arc segment and the intermediate point on the segment. Let *CHORD2* be the line connecting the intermediate point with the end point of this arc segment. Then the center of the circular arc segment is located at the intersection of the perpendicular bisectors of *CHORD1* and *CHORD2*.

   b) If the start, intermediate, and end points of an arc segment are collinear, then the resultant arc segment degenerates to a straight line for which center and radius are not defined. In this case, the circular arc segment is the locus of points defined by the straight line connecting the start and end points.

9) If the cardinality of the attribute *ST_PrivatePoints* is three, then the *ST_CircularString* value is considered a *circular arc*.

10) If an *ST_CircularString* value is simple and closed, then it is considered a *circular ring*.

11) An *ST_CircularString* value returned by the constructor function corresponds to the empty set.

12) An *ST_CircularString* value with the cardinality of the *ST_PrivatePoints* attribute equal to 0 (zero) corresponds to the empty set.

13) For the control point, bulge, bulge normal representation of an *ST_CircularString:*

    a) the *ST_CircularString* value consists of one or more circular arc segments connected end to end. The first segment is defined by two points. The first point is the start point of the arc segment. The second point is the end point of the arc segment. Subsequent segments are defined by their end points only, as the start point is implicitly defined as the previous segment's end point. A mid-arc point is not needed because the bulge defines the shape of the segment.

    b) the cardinalities of the bulge and bulge normal arrays are one less than the cardinality of the control points array.

14) The center point, radius, start angle, end angle representation of an *ST_CircularString* is only valid for circular strings having a single segment and only for 2D. The angles are measured positive from the first coordinate axis in a counterclockwise direction.

15) The contained portion of the circular curve consists of all angles "between" the start angle and end angle in the simple numerical way. If the start angle is less than the end angle, the curve is swept in a counterclockwise direction from the start angle up to the end angle. If the start angle is greater than the end angle, the curve is swept in a clockwise direction from the start angle down to the end angle. Reversing the order of start angle and end angle traces the same curve in the opposite direction. Either or both angles may be negative or larger than 360 to obtain proper control of arcs that wrap around the positive or negative axis. For example, the four possible cases with 0 and 90 degrees are as follows:

    a) start = 0, end = 90: sweeps counterclockwise through 90 degrees from 0 up to 90,

    b) start = 90, end = 0: sweeps clockwise through 90 degrees from 90 down to 0,

    c) start = 90, end = 360: sweeps counterclockwise through 270 degrees from 90 up to 360,

    d) start = 360, end = 90: sweeps clockwise through 270 degrees from 360 down to 90.

7.3.2 ST_CircularString Methods

Purpose

Return an ST_CircularString value constructed from either the well-known text representation; the well-known binary representation; a GML representation; the specified ST_Point values; the specified ST_Point control point, DOUBLE PRECISION bulge and ST_Vector bulge normal ARRAY values; or the specified ST_Point control (center) point, DOUBLE PRECISION radius and ST_Angle start and end angle values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_CircularString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN NEW ST_CircularString(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_CircularString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CircularString
   FOR ST_CircularString
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_CircularString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN NEW ST_CircularString(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_CircularString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN ST_CircularFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN SELF.ST_SRID(0).ST_Points(apointarray)

CREATE CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN SELF.ST_SRID(ansrid).ST_Points(apointarray)

CREATE CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    abulgearray DOUBLE PRECISION
     ARRAY[ST_MaxDoublePrecisionArrayElements],
    anormalarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN NEW ST_CircularString(apointarray, abulgearray, anormalarray, 0)
```

```
CREATE CONSTRUCTOR METHOD ST_CircularString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    abulgearray DOUBLE PRECISION
     ARRAY[ST_MaxDoublePrecisionArrayElements],
    anormalarray ST_Vector ARRAY[ST_MaxVectorArrayElements],
    ansrid INTEGER)
   RETURNS ST_CircularString
   FOR ST_CircularString
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_CircularString
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle)
   RETURNS ST_CircularString
   FOR ST_CircularString
   RETURN NEW ST_CircularString(acenterpoint, aradius, astartangle,
    anendangle, 0)

CREATE CONSTRUCTOR METHOD ST_CircularString
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    ansrid INTEGER)
   RETURNS ST_CircularString
   FOR ST_CircularString
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxDoublePrecisionArrayElements* is the implementation-defined maximum cardinality of an array of DOUBLE PRECISION values.

3) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

4) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

5) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_CircularString(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_CircularString(awktorgml, 0)*.

3) The method *ST_CircularString(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a CHARACTER LARGE OBJECT value *awktorgml*,

b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_CircularString(CHARACTER LARGE OBJECT, INTEGER)*:

Case:

a) If *awktorgml* contains an Arc XML element in the GML representation, then return the result of the value expression: *ST_CircularFromGML(awktorgml, ansrid)*.

b) If *awktorgml* contains an ArcString XML element in the GML representation, then return the result of the value expression: *ST_CircularFromGML(awktorgml, ansrid)*.

c) If *awktorgml* contains an ArcByBulge XML element in the GML representation, then return the result of the value expression: *ST_CircularFromGML(awktorgml, ansrid)*.

d) If *awktorgml* contains an ArcStringByBulge XML element in the GML representation, then return the result of the value expression: *ST_CircularFromGML(awktorgml, ansrid)*.

e) If *awktorgml* contains an ArcByCenterPoint XML element in the GML representation, then return the result of the value expression: *ST_CircularFromGML(awktorgml, ansrid)*.

f) Otherwise, return the result of the value expression: *ST_CircularFromTxt(awktorgml, ansrid)*.

5) The method *ST_CircularString(BINARY LARGE OBJECT)* takes the following input parameter:

a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_CircularString(awkb, 0)*.

7) The method *ST_CircularString(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a BINARY LARGE OBJECT value *awkb*,

b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_CircularFromWKB(awkb, ansrid)*.

9) The method *ST_CircularString(ST_Point ARRAY)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray.*

10) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(ST_Point ARRAY)* returns an *ST_CircularString* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_Points(ST_Point ARRAY)*:

i) the *ST_PrivateDimension* attribute set to 1 (one).

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

v) the *ST_PrivatePoints* attribute set to *apointarray*.

11) The method *ST_CircularString(ST_Point ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(ST_Point ARRAY, INTEGER)* returns an *ST_CircularString* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Points(ST_Point ARRAY)*:

i) the *ST_PrivateDimension* attribute set to 1 (one).

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

v) the *ST_PrivatePoints* attribute set to *apointarray*.

13) The method *ST_CircularString(ST_Point ARRAY, DOUBLE PRECISION ARRAY, ST_Vector ARRAY)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray,*

b) a DOUBLE PRECISION ARRAY value *abulgearray,*

c) an *ST_Vector* ARRAY value *anormalarray.*

14) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(ST_Point ARRAY, DOUBLE PRECISION ARRAY, ST_Vector ARRAY)* returns the result of the value expression: *NEW ST_CircularString(apointarray, abulgearray, anormalarray*, 0*)*.

15) The method *ST_CircularString(ST_Point ARRAY, DOUBLE PRECISION ARRAY, ST_Vector ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray*,

b) a DOUBLE PRECISION ARRAY value *abulgearray,*

c) an *ST_Vector* ARRAY value *anormalarray,*

d) an INTEGER value *ansrid*.

16) Let *PA1* be an *ST_Point ARRAY* containing the *ST_Point* values necessary to construct an *ST_CircularString* value using the constructor method *ST_CircularString(PA1)*. The *PA1 ST_Point* values can be derived from *apointarray, abulgearray* and *anormalarray*.

17) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(ST_Point ARRAY, DOUBLE PRECISION ARRAY, ST_Vector ARRAY, INTEGER)* returns an *ST_CircularString* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Points(ST_Point ARRAY):*

i) the *ST_PrivateDimension* attribute set to 1 (one).

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

v) the *ST_PrivatePoints* attribute set to *apointarray*.

18) The method *ST_CircularString(ST_Point, DOUBLE PRECISION, ST_Angle, ST_Angle)* takes the following input parameters:

a) an *ST_Point* value *acenterpoint,*

b) a DOUBLE PRECISION value *aradius,*

c) an ST_Angle value astartangle,

d) an ST_Angle value anendangle.

19) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(ST_Point, DOUBLE PRECISION, ST_Angle, ST_Angle)* returns the result of the value expression: *NEW ST_CircularString(acenterpoint, aradius, astartangle, anendangle,* 0*)*.

20) The method *ST_CircularString(ST_Point, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER)* takes the following input parameters:

    a) an *ST_Point* value *acenterpoint,*

    b) a DOUBLE PRECISION value *aradius,*

    c) an *ST_Angle* value *astartangle,*

    d) an *ST_Angle* value *anendangle,*

    e) an INTEGER value *ansrid*.

21) Let *PA2* be an *ST_Point* ARRAY containing the *ST_Point* values necessary to construct an *ST_CircularString* value using the constructor method *ST_CircularString(PA2)*. The *PA2 ST_Point* values can be derived from acenterpoint, aradius, astartangle and anendangle.

22) The null-call type-preserving SQL-invoked constructor method *ST_CircularString(ST_Point, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER)* returns an *ST_CircularString* value with:

    a) The spatial reference system identifier set to *ansrid*.

    b) Using the method *ST_Points(ST_Point ARRAY):*

        i) the *ST_PrivateDimension* attribute set to 1 (one).

        ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

        iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

        iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

        v) the *ST_PrivatePoints* attribute set to *apointarray*.

7.3.3 ST_Points Methods

Purpose

Observe and mutate the attribute ST_PrivatePoints of an ST_CircularString value.

Definition

```
CREATE METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_CircularString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivatePoints
      END

CREATE METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CircularString
   FOR ST_CircularString
   BEGIN
      DECLARE counter INTEGER;
      -- If apointarray is the null value, contains null elements, or
      -- contains consecutive duplicate points, then raise an exception.
      CALL ST_CheckConsecDups(apointarray);
      -- If any segment start and end point are not distinct, then raise
      -- an exception.
      SET counter = 3;
      WHILE counter <= CARDINALITY(apointarray) DO
         -- If the current element (a segment end point) is equal to the
         -- same segment’s start point, then raise an exception.
         IF SELF.ST_Is3D = 0 AND
            apointarray[counter].ST_Equals(apointarray[counter-2]) = 1
            THEN
            SIGNAL SQLSTATE '2FF05'
               SET MESSAGE_TEXT = 'duplicate value';
         IF SELF.ST_Is3D = 1 AND
            apointarray[counter].ST_3DEquals(apointarray[counter-2]) = 1
            THEN
            SIGNAL SQLSTATE '2FF05'
               SET MESSAGE_TEXT = 'duplicate value';
         END IF;
         SET counter = counter + 2;
      END WHILE;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_CircularString);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and apointarray.
      IF (CARDINALITY(apointarray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(apointarray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      RETURN
         SELF.ST_PrivateDimension(1).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(apointarray)).
            ST_PrivateIs3D(ST_GetIs3D(apointarray)).
```

```
            ST_PrivateIsMeasured(ST_GetIsMeasured(apointarray)).
            ST_PrivatePoints(apointarray);
      END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Points()* has no input parameters.
2) For the null-call method *ST_Points()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivatePoints* of SELF.
3) The method *ST_Points(ST_Point ARRAY)* takes the following input parameters:
   a) an *ST_Point* ARRAY value *apointarray*.
4) For the type-preserving method *ST_Points(ST_Point ARRAY)*:
   a) Call the procedure *ST_CheckConsecDups(ST_Geometry ARRAY)* to check if *apointarray* is the null value or contains null elements.
   b) If the end point of any segment is spatially equal to the start point of that segment, then an exception condition is raised: SQL/MM Spatial exception – duplicate value.
   c) Case:
      i) If SELF is the null value, then return the null value.
      ii) If the cardinality of *apointarray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID*(*apointarray*), then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.
      iii) Otherwise, return an *ST_CircularString* value with:
         1) the dimension set to 1 (one).
         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(apointarray)*.
         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.
         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.
         5) the attribute *ST_PrivatePoints* set to *apointarray*.

7.3.4 ST_NumPoints Method

Purpose

Return the cardinality of the ST_PrivatePoints attribute of an ST_CircularString value.

Definition

```
CREATE METHOD ST_NumPoints()
   RETURNS INTEGER
   FOR ST_CircularString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivatePoints)
      END
```

Description

1) The method *ST_NumPoints()* has no input parameters.

2) For the null-call method *ST_NumPoints()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the cardinality of the *ST_PrivatePoints* attribute.

7.3.5 ST_PointN Method

Purpose

Return the specified element in the ST_PrivatePoints attribute of an ST_CircularString value.

Definition

```
CREATE METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   FOR ST_CircularString
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Point);
      END IF;
      IF aposition < 1 OR
         aposition > SELF.ST_NumPoints() THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Point);
         END;
      END IF;
      RETURN SELF.ST_PrivatePoints[aposition];
   END
```

Description

1) The method *ST_PointN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_PointN(INTEGER)*:

   Case:
   a) If SELF is an empty set, then return the null value.
   b) If *aposition* is less than 1 (one) or greater than the cardinality of the attribute *ST_PrivatePoints*, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return an *ST_Point* value at element *aposition* in the attribute *ST_PrivatePoints* of SELF.

7.3.6 ST_NumSegments Method

Purpose

Return the number of curve segments (arcs) of an ST_CircularString value.

Definition

```
CREATE METHOD ST_NumSegments()
   RETURNS INTEGER
   FOR ST_CircularString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            (CARDINALITY(SELF.ST_PrivatePoints) – 1) / 2
      END
```

Description

1) The method *ST_NumSegments()* has no input parameters.
2) For the null-call method *ST_NumSegments()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise,
      i) Let *C* equal the cardinality of the *ST_PrivatePoints* attribute.
      ii) Return the result of (*C* - 1) / 2.

7.3.7 ST_SegmentN Method

Purpose

Return the Nth curve segment of an ST_CircularString value as an ST_CircularString having a single curve segment.

Definition

```
CREATE METHOD ST_SegmentN
   (aposition INTEGER)
   RETURNS ST_CircularString
   FOR ST_CircularString
   BEGIN
      DECLARE astartpointn INTEGER;
      DECLARE ansrid INTEGER;
      DECLARE apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements];
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_CircularString);
      END IF;
      IF aposition < 1 OR
         aposition > SELF.ST_NumSegments() THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_CircularString);
         END;
      END IF;
      SET astartpointn = (aposition * 2) – 1;
      SET ansrid = SELF.ST_SRID();
      -- create apointarray of three elements, starting with the ST_Point
      at position astartpointn in SELF.ST_PrivatePoints
      -- Set apointarray to an empty array.
      SET apointarray = CAST(ARRAY[] AS
         ST_Point ARRAY[ST_MaxGeometryArrayElements]);
      SET apointarray = apointarray || SELF.ST_PointN(astartpointn);
      SET apointarray = apointarray || SELF.ST_PointN(astartpointn+1);
      SET apointarray = apointarray || SELF.ST_PointN(astartpointn+2);
      RETURN NEW ST_CircularString(apointarray,ansrid);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_SegmentN(INTEGER)* takes the following input parameters:

   a) an INTEGER value *aposition*.

2) For the null-call method *ST_SegmentN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If *aposition* is less than 1 (one) or greater than the number of segments in SELF, then:

      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.

      ii) Return the null value.

   c) Otherwise, return an *ST_CircularString* value constructed from the *aposition* segment of SELF.

7.3.8 ST_MidPointRep Method

Purpose

Return an ST_Point ARRAY which uniquely identifies an ST_CircularString value, including the start, mid, and end points of each curve segment.

Definition

```
CREATE METHOD ST_MidPointRep()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_CircularString
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_MidPointRep()* has no input parameters.
2) For the null-call method *ST_MidPointRep()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return an *ST_Points* ARRAY such that:
      i) For the first circular arc segment of the curve, the points returned are the start, mid, and end points of the segment.
      ii) For all subsequent segments, the points returned are the mid and end points of the respective segment.
3) If *SELF.ST_Is3D()* is equal to 1 (one), then:
   a) The z coordinate values are considered in the calculation.
   b) The *ST_Point* values include the z coordinate value.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
   a) The m coordinate values are considered in the calculation.
   b) The *ST_Point* values include the m coordinate value.
5) The spatial reference system identifier of the returned *ST_Point* values is equal to the spatial reference system identifier of SELF.

7.3.9 ST_Bulge Method

Purpose

Return the DOUBLE PRECISION value that is the bulge of an ST_CircularString value having a single curve segment.

Definition

```
CREATE METHOD ST_Bulge()
   RETURNS DOUBLE PRECISION
   FOR ST_CircularString
   BEGIN
      DECLARE abulge DOUBLE PRECISION;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate abulge as the bulge of SELF
      --
      -- See Description
      --
      RETURN abulge;
   END
```

Description

1) The method *ST_Bulge()* has no input parameters.
2) For the null-call method *ST_Bulge()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.
   c) Otherwise, return the DOUBLE PRECISION bulge value as defined in ISO 19107:2003.

7.3.10 ST_BulgeNormal Method

Purpose

Return the ST_Vector value that is the bulge normal of an ST_CircularString value having a single curve segment.

Definition

```
CREATE METHOD ST_BulgeNormal()
   RETURNS ST_Vector
   FOR ST_CircularString
   BEGIN
      DECLARE abulgenormal ST_Vector;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Vector);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate abulgenormal as the bulge normal of SELF
      --
      -- See Description
      --
      RETURN abulgenormal;
   END
```

Description

1) The method *ST_BulgeNormal()* has no input parameters.

2) For the null-call method *ST_BulgeNormal()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.

   c) Otherwise, return the *ST_Vector* bulge normal value as defined in ISO IS 19107:2003, 6.4.17.4, "normal".

7.3.11 ST_Center Method

Purpose

Return the ST_Point value that is the center of the circle of which an ST_CircularString value having a single curve segment is a portion.

Definition

```
CREATE METHOD ST_Center()
   RETURNS ST_Point
   FOR ST_CircularString
   BEGIN
      DECLARE apoint ST_Point;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Point);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate apoint as the center point of SELF
      --
      -- See Description
      --
      RETURN apoint;
   END
```

Description

1) The method *ST_Center()* has no input parameters.

2) For the null-call method *ST_Center()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.

   c) Otherwise, return the *ST_Point* value that is the center of the circle of which SELF is a portion, including z (but not m) coordinate values if appropriate, and having the same spatial reference system as SELF.

7.3.12 ST_Radius Method

Purpose

Return the DOUBLE PRECISION value that is the radius of the circle of which an ST_CircularString value having a single curve segment is a portion.

Definition

```
CREATE METHOD ST_Radius()
   RETURNS DOUBLE PRECISION
   FOR ST_CircularString
   BEGIN
      DECLARE aradius DOUBLE PRECISION;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate aradius as the radius of SELF
      --
      -- See Description
      --
      RETURN aradius;
   END

CREATE METHOD ST_Radius
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_CircularString
   BEGIN
      DECLARE aradius DOUBLE PRECISION;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate aradius as the radius of SELF
      --
      -- See Description
      --
      RETURN aradius;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Radius()* has no input parameters.

2) For the null-call method *ST_Radius()*:

   a) Case:

      i) If SELF is an empty set, then return the null value.

      ii) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.

iii) Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Radius()* is in the linear unit of measure identified by <linear unit>.

ii) Otherwise, the value returned by *ST_Radius()* is in an implementation-defined unit of measure.

3) The method *ST_Radius(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_Radius(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the radius of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.

iii) Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

7.3.13 ST_StartAngle Method

Purpose

Return the ST_Angle value that is the start angle of an ST_CircularString value having a single curve segment.

Definition

```
CREATE METHOD ST_StartAngle()
   RETURNS ST_Angle
   FOR ST_CircularString
   BEGIN
      DECLARE anangle ST_Angle;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Angle);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate anangle as the start angle of SELF
      --
      -- See Description
      --
      RETURN anangle;
   END
```

Description

1) The method *ST_StartAngle()* has no input parameters.

2) For the null-call method *ST_StartAngle()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.

   c) Otherwise, return the *ST_Angle* value that is the start angle of SELF, such that z (but not m) coordinate values are considered in the calculation.

7.3.14 ST_EndAngle Method

Purpose

Return the ST_Angle value that is the end angle of an ST_CircularString value having a single curve segment.

Definition

```
CREATE METHOD ST_EndAngle()
   RETURNS ST_Angle
   FOR ST_CircularString
   BEGIN
      DECLARE anangle ST_Angle;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Angle);
      END IF;
      IF SELF.ST_NumSegments > 1 THEN
         SIGNAL SQLSTATE '2FF76'
            SET MESSAGE_TEXT = 'curve has multiple segments';
      END IF;
      -- calculate anangle as the end angle of SELF
      --
      -- See Description
      --
      RETURN anangle;
   END
```

Description

1) The method *ST_EndAngle()* has no input parameters.

2) For the null-call method *ST_EndAngle()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If SELF has more than one segment, then an exception condition is raised: *SQL/MM Spatial exception – curve has multiple segments*.

   c) Otherwise, return the *ST_Angle* value that is the end angle of SELF, such that z (but not m) coordinate values are considered in the calculation.

7.3.15 ST_StartPoint Method

Purpose

Return the start point of an ST_CircularString value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_CircularString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[1]
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.
2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[1]*.

7.3.16 ST_EndPoint Method

Purpose

Return the end point of an ST_CircularString value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_CircularString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[SELF.ST_NumPoints()]
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.

2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[SELF.ST_NumPoints()]*.

7.3.17 ST_CircularFromTxt Functions

Purpose

Return an ST_CircularString value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CircularString value.

Definition

```
CREATE FUNCTION ST_CircularFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CircularFromTxt(awkt, 0)

CREATE FUNCTION ST_CircularFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_CircularFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_CircularFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CircularFromTxt(awkt, 0)*.
3) The function *ST_CircularFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CircularFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_CircularString* value.

      If *awkt* is not producible in the BNF for <circularstring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_CircularString)*.

7.3.18 ST_CircularFromWKB Functions

Purpose

Return an ST_CircularString value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CircularString value.

Definition

```
CREATE FUNCTION ST_CircularFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CircularFromWKB(awkb, 0)

CREATE FUNCTION ST_CircularFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_CircularFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_CircularFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_CircularFromWKB(awkb, 0)*.
3) The function *ST_CircularFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CircularFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_CircularString* value.

      If *awkb* is not producible in the BNF for <circularstring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_CircularString)*.

7.3.19 ST_CircularFromGML Functions

Purpose

Return an ST_CircularString value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Arc, ArcString, ArcByBulge, ArcStringByBulge or ArcByCenterPoint representation of an ST_CircularString value.

Definition

```
CREATE FUNCTION ST_CircularFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CircularFromGML(agml, 0)

CREATE FUNCTION ST_CircularFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_CircularString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_CircularFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_CircularFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CircularFromGML(agml, 0)*.
3) The function *ST_CircularFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CircularFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain an Arc, ArcString, ArcByBulge, ArcStringByBulge or ArcByCenterPoint XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_CircularString)*.

7.4 ST_Circle Type and Routines

7.4.1 ST_Circle Type

Purpose

The ST_Circle type is a subtype of the ST_Curve type and represents a curve with circular interpolation having a single arc which is simple and closed.

Definition

```
CREATE TYPE ST_Circle
   UNDER ST_Curve
   AS (
      ST_PrivatePoints ST_Point
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Circle
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_Circle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Circle
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_Circle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Circle
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_Circle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Circle
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_Circle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Circle
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Circle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Circle
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_Circle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Circle
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    anormalvector ST_Vector)
   RETURNS ST_Circle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Circle
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    anormalvector ST_Vector,
    ansrid INTEGER)
   RETURNS ST_Circle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Circle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Radius()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Radius
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Center()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Normal()
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
5) The attribute *ST_PrivatePoints* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePoints*.

Description

1) The *ST_Circle* type provides for public use:
   a) a method *ST_Circle(CHARACTER LARGE OBJECT)*,

b) a method *ST_Circle(CHARACTER LARGE OBJECT, INTEGER)*,

c) a method *ST_Circle(BINARY LARGE OBJECT)*,

d) a method *ST_Circle(BINARY LARGE OBJECT, INTEGER)*,

e) a method *ST_Circle(ST_Point ARRAY)*,

f) a method *ST_Circle(ST_Point ARRAY, INTEGER)*,

g) a method *ST_Circle(ST_Point, DOUBLE PRECISION, ST_Vector)*,

h) a method *ST_Circle(ST_Point, DOUBLE PRECISION, ST_Vector, INTEGER)*,

i) a method *ST_Points()*,

j) a method *ST_Points(ST_Point ARRAY)*,

k) a method *ST_PointN(INTEGER)*,

l) a method *ST_Radius()*,

m) a method *ST_Radius(CHARACTER VARYING)*,

n) a method *ST_Center()*,

o) a method *ST_Normal()*,

p) an overriding method *ST_StartPoint()*,

q) an overriding method *ST_EndPoint()*,

r) a function *ST_CircleFromTxt(CHARACTER LARGE OBJECT)*,

s) a function *ST_CircleFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,

t) a function *ST_CircleFromWKB(BINARY LARGE OBJECT)*,

u) a function *ST_CircleFromWKB(BINARY LARGE OBJECT, INTEGER)*,

v) a function *ST_CircleFromGML(CHARACTER LARGE OBJECT)*,

w) a function *ST_CircleFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivatePoints* attribute contains the collection of *ST_Point* values.

3) The *ST_PrivatePoints* attribute shall not be the null value. The elements in the *ST_PrivatePoints* attribute shall not be the null value.

4) All the *ST_Point* values in the *ST_PrivatePoints* attribute shall be in the same spatial reference system as the *ST_Circle* value.

5) The coordinate dimension of an *ST_Circle* value is the number of coordinate values associated with the *ST_Point* values.

6) An *ST_Circle* value consists of one circular arc segment of constant radius, defined by three unique, non-colinear points. The first point is the start point of the arc segment. The arc passes through the second and third control points. The arc is then extended past the third control point until the first control point is encountered.

7) The circular arc segment is the locus of points defined as follows:

   a) The circular arc segment is the locus of points a distance *R* from the center of the arc, beginning at the start point, passing through the second and third points, and ending at the start point of the circular arc segment. The distance *R* is the radius of the circular arc segment, and is equal to the distance from the center of the circular arc segment and the three control points. The center of the circular arc segment is defined as follows:

   b) Let *CHORD1* be the line connecting the start point of a circular arc segment and the second control point on the segment. Let *CHORD2* be the line connecting the second control point with the third control point of this arc segment. Then the center of the circular arc segment is located at the intersection of the perpendicular bisectors of *CHORD1* and *CHORD2*.

8) The cardinality of the attribute *ST_PrivatePoints* is three.

9) An *ST_Circle* value is simple and closed, and is considered a *circular ring*.
10) An *ST_Circle* value returned by the constructor function corresponds to the empty set.
11) An *ST_Circle* value with the cardinality of the *ST_PrivatePoints* attribute equal to 0 (zero) corresponds to the empty set.

7.4.2 ST_Circle Methods

Purpose

Return an ST_Circle value constructed from either the well-known text representation, the well-known binary representation, a GML representation, the specified ST_Point values, or the specified center point, radius and normal vector.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Circle
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Circle
   FOR ST_Circle
   RETURN NEW ST_Circle(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_Circle
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Circle
   FOR ST_Circle
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Circle
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Circle
   FOR ST_Circle
   RETURN NEW ST_Circle(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_Circle
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Circle
   FOR ST_Circle
   RETURN ST_CircleFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_Circle
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Circle
   FOR ST_Circle
   RETURN SELF.ST_SRID(0).ST_Points(apointarray)

CREATE CONSTRUCTOR METHOD ST_Circle
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_Circle
   FOR ST_Circle
   RETURN SELF.ST_SRID(ansrid).ST_Points(apointarray)

CREATE CONSTRUCTOR METHOD ST_Circle
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    anormalvector ST_Vector)
   RETURNS ST_Circle
   FOR ST_Circle
   RETURN NEW ST_Circle(acenterpoint, aradius, anormalvector, 0)
```

```
CREATE CONSTRUCTOR METHOD ST_Circle
   (acenterpoint ST_Point,
    aradius DOUBLE PRECISION,
    anormalvector ST_Vector,
    ansrid INTEGER)
   RETURNS ST_Circle
   FOR ST_Circle
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Geometry* value.

Description

1) The method *ST_Circle(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_Circle(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_Circle(awktorgml, 0)*.
3) The method *ST_Circle(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_Circle(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If *awktorgml* contains a Circle XML element in the GML representation, then return the result of the value expression: *ST_CircleFromGML(awktorgml, ansrid)*.
   b) If *awktorgml* contains a CircleByCenterPoint XML element in the GML representation, then return the result of the value expression: *ST_CircleFromGML(awktorgml, ansrid)*.
   c) Otherwise, return the result of the value expression: *ST_CircleFromTxt(awktorgml, ansrid)*.
5) The method *ST_Circle(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_Circle(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_Circle(awkb, 0)*.
7) The method *ST_Circle(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
8) The null-call type-preserving SQL-invoked constructor method *ST_Circle(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_CircleFromWKB(awkb, ansrid)*.
9) The method *ST_Circle(ST_Point ARRAY)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray.*

10) The null-call type-preserving SQL-invoked constructor method *ST_Circle(ST_Point ARRAY)* returns an *ST_Circle* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_Points(ST_Point ARRAY):*

i) the *ST_PrivateDimension* attribute set to 1 (one).

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

v) the *ST_PrivatePoints* attribute set to *apointarray*.

11) The method *ST_Circle(ST_Point ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_Circle(ST_Point ARRAY, INTEGER)* returns an *ST_Circle* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Points(ST_Point ARRAY):*

i) the *ST_PrivateDimension* attribute set to 1 (one).

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

v) the *ST_PrivatePoints* attribute set to *apointarray*.

13) The method *ST_Circle(ST_Point, DOUBLE PRECISION, ST_Vector)* takes the following input parameters:

a) an *ST_Point* value *acenterpoint,*

b) a DOUBLE PRECISION value *aradius,*

c) an *ST_Vector* value *anormalvector.*

14) The null-call type-preserving SQL-invoked constructor method *ST_Circle(ST_Point, DOUBLE PRECISION, ST_Vector*) returns the result of the value expression: *NEW ST_Circle(acenterpoint, aradius, anormalvector, 0)*.

15) The method *ST_Circle(ST_Point, DOUBLE PRECISION, ST_Vector)* takes the following input parameters:

a) an *ST_Point* value *acenterpoint,*

b) a DOUBLE PRECISION value *aradius,*

c) an *ST_Vector* value *anormalvector,*

d) an INTEGER value *ansrid*.

16) Let *PA1* be an *ST_Point* ARRAY containing the *ST_Point* values necessary to construct an *ST_Circle* value using the constructor method *ST_Circle(PA1)*. The *PA1 ST_Point* values can be derived from *acenterpoint, aradius* and *anormalvector*.

17) The null-call type-preserving SQL-invoked constructor method *ST_Circle(ST_Point, DOUBLE PRECISION, ST_Vector, INTEGER)* returns an *ST_Circle* value with:

    a) The spatial reference system identifier set to *ansrid*.

    b) Using the method *ST_Points(ST_Point ARRAY):*

        i) the *ST_PrivateDimension* attribute set to 1 (one).

        ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

        iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

        iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

        v) the *ST_PrivatePoints* attribute set to *apointarray*.

7.4.3 ST_Points Methods

Purpose

Observe and mutate the attribute ST_PrivatePoints of an ST_Circle value.

Definition

```
CREATE METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_Circle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivatePoints
   END

CREATE METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Circle
   FOR ST_Circle
   BEGIN
      DECLARE anotherpointarray ST_Point
       ARRAY[ST_MaxGeometryArrayElements];
      DECLARE ansrid INTEGER;
      DECLARE alinestring ST_LineString
      -- If apointarray is the null value, contains null elements, or
      -- contains consecutive duplicate points, then raise an exception.
      CALL ST_CheckConsecDups(apointarray);
      -- Check that apointarray has exactly three points
      IF CARDINALITY(apointarray) <> 3 THEN
         SIGNAL SQLSTATE '2FF77'
            SET MESSAGE_TEXT = 'exactly three points are required';
      -- Check that last point does not equal first point
      IF apointarray[3] = apointarray[1] THEN
         SIGNAL SQLSTATE '2FF05'
            SET MESSAGE_TEXT = 'duplicate value';
      -- Check that points are not collinear
      SET ansrid = ST_CheckSRID(apointarray);
      -- create anotherpointarray of two elements, the first and third
      elements from apointarray
      -- Set anotherpointarray to an empty array.
      SET anotherpointarray = CAST(ARRAY[] AS
         ST_Point ARRAY[ST_MaxGeometryArrayElements]);
      SET anotherpointarray = anotherpointarray || apointarray[1];
      SET anotherpointarray = anotherpointarray || apointarray[3];
      -- construct a linestring connecting the first and last points
      SET alinestring = NEW ST_LineString(anotherpointarray, ansrid);
      IF apointarray[2].ST_Intersects(alinestring) THEN
         SIGNAL SQLSTATE '2FF78'
            SET MESSAGE_TEXT = 'points are collinear';
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Circle);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and apointarray.
      IF (CARDINALITY(apointarray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(apointarray)) THEN
         SIGNAL SQLSTATE '2FF10'
```

```
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
        END IF;
        RETURN
          SELF.ST_PrivateDimension(1).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(apointarray)).
            ST_PrivateIs3D(ST_GetIs3D(apointarray)).
            ST_PrivateIsMeasured(ST_GetIsMeasured(apointarray)).
            ST_PrivatePoints(apointarray);
    END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Points()* has no input parameters.

2) For the null-call method *ST_Points()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivatePoints* of SELF.

3) The method *ST_Points(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*.

4) For the type-preserving method *ST_Points(ST_Point ARRAY)*:

   a) Call the procedure *ST_CheckConsecDups(ST_Geometry ARRAY)* to check if *apointarray* is the null value or contains null elements.

   b) Case:

      i) If the cardinality of *apointarray* is not 3, then an exception condition is raised: *SQL/MM Spatial exception – exactly three points are required*.

      ii) If the last ST_Point value in *apointarray* is equal to the first ST_Point value, then an exception condition is raised: *SQL/MM Spatial exception – duplicate value*.

      iii) If the three ST_Point values in *apointarray* are collinear, then an exception condition is raised: *SQL/MM Spatial exception – points are collinear*.

      iv) If SELF is the null value, then return the null value.

      v) If the cardinality of *apointarray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID*(*apointarray*), then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

      vi) Otherwise, return an *ST_Circle* value with:

         1) the dimension set to 1 (one).

         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(apointarray)*.

         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

         5) the attribute *ST_PrivatePoints* set to *apointarray*.

7.4.4 ST_PointN Method

Purpose

Return the specified element in the ST_PrivatePoints attribute of an ST_Circle value.

Definition

```
CREATE METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   FOR ST_Circle
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Point);
      END IF;
      IF aposition < 1 OR
         aposition > 3 THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Point);
         END;
      END IF;
      RETURN SELF.ST_PrivatePoints[aposition];
   END
```

Description

1) The method *ST_PointN(INTEGER)* takes the following input parameters:

   a) an INTEGER value *aposition*.

2) For the null-call method *ST_PointN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If *aposition* is less than 1 (one) or greater than 3, then:

      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.

      ii) Return the null value.

   c) Otherwise, return an *ST_Point* value at element *aposition* in the attribute *ST_PrivatePoints* of SELF.

7.4.5 ST_Radius Method

Purpose

Return the DOUBLE PRECISION value that is the radius of the circle.

Definition

```
CREATE METHOD ST_Radius()
   RETURNS DOUBLE PRECISION
   FOR ST_Circle
   BEGIN
      DECLARE aradius DOUBLE PRECISION;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      -- calculate aradius as the radius of SELF
      --
      -- See Description
      --
      RETURN aradius;
   END

CREATE METHOD ST_Radius
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Circle
   BEGIN
      DECLARE aradius DOUBLE PRECISION;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      -- calculate aradius as the radius of SELF
      --
      -- See Description
      --
      RETURN aradius;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Radius()* has no input parameters.

2) For the null-call method *ST_Radius()*:

   a) Case:

      i) If SELF is an empty set, then return the null value.

      ii) Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

   b) Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Radius()* is in the linear unit of measure identified by <linear unit>.

      ii) Otherwise, the value returned by *ST_Radius()* is in an implementation-defined unit of measure.

3) The method *ST_Radius(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_Radius(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the radius of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
   d) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.
   e) The returned value is in the units indicated by *aunit*.

7.4.6 ST_Center Method

Purpose

Return the ST_Point value that is the center of the circle.

Definition

```
CREATE METHOD ST_Center()
   RETURNS ST_Point
   FOR ST_Circle
   BEGIN
      DECLARE apoint ST_Point;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Point);
      END IF;
      -- calculate apoint as the center point of SELF
      --
      -- See Description
      --
      RETURN apoint;
   END
```

Description

1) The method *ST_Center()* has no input parameters.

2) For the null-call method *ST_Center()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_Point* value that is the center of SELF, including z (but not m) coordinate values if appropriate, and having the same spatial reference system as SELF.

7.4.7 ST_Normal Method

Purpose

Return the ST_Vector value that is the normal of the circle.

Definition

```
CREATE METHOD ST_Normal()
   RETURNS ST_Vector
   FOR ST_Circle
   BEGIN
      DECLARE avector ST_Vector;
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Vector);
      END IF;
      -- calculate avector as the normal of SELF
      --
      -- See Description
      --
      RETURN avector;
   END
```

Description

1) The method *ST_Normal()* has no input parameters.

2) For the null-call method *ST_Normal()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_Vector* value that is the normal of SELF, including z (but not m) coordinate values if appropriate, and having the same spatial reference system as SELF.

7.4.8 ST_StartPoint Method

Purpose

Return the start point of an ST_Circle value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_Circle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[1]
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.
2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[1]*.

7.4.9 ST_EndPoint Method

Purpose

Return the end point of an ST_Circle value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_Circle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[1]
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.
2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[1]*.

7.4.10 ST_CircleFromTxt Functions

Purpose

Return an ST_Circle value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Circle value.

Definition

```
CREATE FUNCTION ST_CircleFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CircleFromTxt(awkt, 0)

CREATE FUNCTION ST_CircleFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_CircleFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_CircleFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CircleFromTxt(awkt, 0)*.
3) The function *ST_CircleFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CircleFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_Circle* value.

      If *awkt* is not producible in the BNF for <circle text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_Circle)*.

7.4.11 ST_CircleFromWKB Functions

Purpose

Return an ST_Circle value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Circle value.

Definition

```
CREATE FUNCTION ST_CircleFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CircleFromWKB(awkb, 0)

CREATE FUNCTION ST_CircleFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_CircleFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_CircleFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_CircleFromWKB(awkb, 0)*.
3) The function *ST_CircleFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CircleFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_Circle* value.

      If *awkb* is not producible in the BNF for <circle binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_Circle)*.

7.4.12 ST_CircleFromGML Functions

Purpose

Return an ST_Circle value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Circle or CircleByCenterPoint representation of an ST_Circle value.

Definition

```
CREATE FUNCTION ST_CircleFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CircleFromGML(agml, 0)
```

```
CREATE FUNCTION ST_CircleFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_Circle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_CircleFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_CircleFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CircleFromGML(agml, 0)*.
3) The function *ST_CircleFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CircleFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a Circle or CircleByCenterPoint XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_Circle)*.

7.5 ST_GeodesicString Type and Routines

7.5.1 ST_GeodesicString Type

Purpose

The ST_GeodesicString type is a subtype of the ST_Curve type and represents a curve with geodesic interpolation.

Definition

```
CREATE TYPE ST_GeodesicString
   UNDER ST_Curve
   AS (
      ST_PrivatePoints ST_Point
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_GeodesicString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_GeodesicString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_GeodesicString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_GeodesicString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_GeodesicString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_GeodesicString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_GeodesicString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_GeodesicString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_GeodesicString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeodesicString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_GeodesicString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeodesicString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumPoints()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivatePoints* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePoints*.

Description

1) The *ST_GeodesicString* type provides for public use:
   a) a method *ST_GeodesicString(CHARACTER LARGE OBJECT)*,
   b) a method *ST_GeodesicString(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_GeodesicString(BINARY LARGE OBJECT)*,
   d) a method *ST_GeodesicString(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_GeodesicString(ST_Point ARRAY)*,
   f) a method *ST_GeodesicString(ST_Point ARRAY, INTEGER)*,
   g) a method *ST_Points()*,
   h) a method *ST_Points(ST_Point ARRAY)*,
   i) a method *ST_NumPoints()*,
   j) a method *ST_PointN(INTEGER)*,
   k) an overriding method *ST_StartPoint()*,
   l) an overriding method *ST_EndPoint()*,
   m) a function *ST_GeodesicFromTxt(CHARACTER LARGE OBJECT)*,
   n) a function *ST_GeodesicFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,
   o) a function *ST_GeodesicFromWKB(BINARY LARGE OBJECT)*,
   p) a function *ST_GeodesicFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   q) a function *ST_GeodesicFromGML(CHARACTER LARGE OBJECT)*,
   r) a function *ST_GeodesicFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The *ST_PrivatePoints* attribute contains the collection of *ST_Point* values.
3) The *ST_PrivatePoints* attribute shall not be the null value. The elements in the *ST_PrivatePoints* attribute shall not be the null value.
4) If the cardinality of the *ST_PrivatePoints* attribute is greater than or equal to two, then the *ST_GeodesicString* value is well formed.
5) All the *ST_Point* values in the *ST_PrivatePoints* attribute shall be in the same spatial reference system as the *ST_GeodesicString* value.
6) The coordinate dimension of an *ST_GeodesicString* value is the number of coordinate values associated with the *ST_Point* values.
7) The type *ST_GeodesicString* is a subtype of *ST_Curve* with geodesic interpolation between control points. Each consecutive pair of control points defines a *geodesic segment*.
8) The control points are a sequence of positions between which the *ST_GeodesicString* is interpolated using geodesics from the geoid or ellipsoid of the coordinate reference system being used.
9) If the cardinality of the *ST_PrivatePoints* attribute is two, then the *ST_GeodesicString* value is called a *geodesic*.
10) If an *ST_GeodesicString* value is simple and closed, then it is called a *geodesic ring*.
11) An *ST_GeodesicString* value returned by the constructor function corresponds to the empty set.

12) An *ST_GeodesicString* value with the cardinality of the *ST_PrivatePoints* attribute equal to 0 (zero) corresponds to the empty set.

7.5.2 ST_GeodesicString Methods

Purpose

Return an ST_GeodesicString value constructed from either the well-known text representation, the well-known binary representation, a GML representation, or the specified ST_Point values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_GeodesicString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   RETURN NEW ST_GeodesicString(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_GeodesicString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_GeodesicString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   RETURN NEW ST_GeodesicString(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_GeodesicString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   RETURN ST_GeodesicFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_GeodesicString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   RETURN SELF.ST_SRID(0).ST_Points(apointarray)

CREATE CONSTRUCTOR METHOD ST_GeodesicString
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   RETURN SELF.ST_SRID(ansrid).ST_Points(apointarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Geometry* value.

Description

1) The method *ST_GeodesicString(CHARACTER LARGE OBJECT)* takes the following input parameter:

a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_GeodesicString(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_GeodesicString(awktorgml, 0)*.

3) The method *ST_GeodesicString(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_GeodesicString(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Geodesic XML element in the GML representation, then return the result of the value expression: *ST_GeodesicFromGML(awktorgml, ansrid)*.

   b) If *awktorgml* contains a GeodesicString XML element in the GML representation, then return the result of the value expression: *ST_GeodesicFromGML(awktorgml, ansrid)*.

   c) Otherwise, return the result of the value expression: *ST_GeodesicFromTxt(awktorgml, ansrid)*.

5) The method *ST_GeodesicString(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_GeodesicString(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_GeodesicString(awkb, 0)*.

7) The method *ST_GeodesicString(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_GeodesicString(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_GeodesicFromWKB(awkb, ansrid)*.

9) The method *ST_GeodesicString(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray.*

10) The null-call type-preserving SQL-invoked constructor method *ST_GeodesicString(ST_Point ARRAY)* returns an *ST_GeodesicString* value with:

   a) The spatial reference system identifier set to 0 (zero).

   b) Using the method *ST_Points(ST_Point ARRAY):*

      i) the *ST_PrivateDimension* attribute set to 1 (one).

      ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

      iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

      iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

      v) the *ST_PrivatePoints* attribute set to *apointarray*.

11) The method *ST_GeodesicString(ST_Point ARRAY, INTEGER)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*,

   b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_GeodesicString(ST_Point ARRAY, INTEGER)* returns an *ST_GeodesicString* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Points(ST_Point ARRAY):*

   i) the *ST_PrivateDimension* attribute set to 1 (one).

   ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

   iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

   iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

   v) the *ST_PrivatePoints* attribute set to *apointarray*.

7.5.3 ST_Points Methods

Purpose

Observe and mutate the attribute ST_PrivatePoints of an ST_GeodesicString value.

Definition

```
CREATE METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_GeodesicString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivatePoints
      END

CREATE METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeodesicString
   FOR ST_GeodesicString
   BEGIN
      -- If apointarray is the null value, contains null elements, or
      -- contains consecutive duplicate points, then raise an exception.
      CALL ST_CheckConsecDups(apointarray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_GeodesicString);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and apointarray.
      IF (CARDINALITY(apointarray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(apointarray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      RETURN
         SELF.ST_PrivateDimension(1).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(apointarray)).
            ST_PrivateIs3D(ST_GetIs3D(apointarray)).
            ST_PrivateIsMeasured(ST_GetIsMeasured(apointarray)).
            ST_PrivatePoints(apointarray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Points()* has no input parameters.

2) For the null-call method *ST_Points()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivatePoints* of SELF.

3) The method *ST_Points(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*.

4) For the type-preserving method *ST_Points(ST_Point ARRAY)*:

   a) Call the procedure *ST_CheckConsecDups(ST_Geometry ARRAY)* to check if *apointarray* is the null value or contains null elements.

   b) Case:

      i) If SELF is the null value, then return the null value.

      ii) If the cardinality of *apointarray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID*(*apointarray*), then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

      iii) Otherwise, return an *ST_GeodesicString* value with:

         1) the dimension set to 1 (one).

         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(apointarray)*.

         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

         5) the attribute *ST_PrivatePoints* set to *apointarray*.

7.5.4 ST_NumPoints Method

Purpose

Return the cardinality of the ST_PrivatePoints attribute of an ST_GeodesicString value.

Definition

```
CREATE METHOD ST_NumPoints()
   RETURNS INTEGER
   FOR ST_GeodesicString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivatePoints)
      END
```

Description

1) The method *ST_NumPoints()* has no input parameters.

2) For the null-call method *ST_NumPoints()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the cardinality of the *ST_PrivatePoints* attribute.

7.5.5 ST_PointN Method

Purpose

Return the specified element in the ST_PrivatePoints attribute of an ST_GeodesicString value.

Definition

```
CREATE METHOD ST_PointN
   (aposition INTEGER)
   RETURNS ST_Point
   FOR ST_GeodesicString
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Point);
      END IF;
      IF aposition < 1 OR
         aposition > SELF.ST_NumPoints() THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Point);
         END;
      END IF;
      RETURN SELF.ST_PrivatePoints[aposition];
   END
```

Description

1) The method *ST_PointN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_PointN(INTEGER)*:

   Case:
   a) If SELF is an empty set, then return the null value.
   b) If *aposition* is less than 1 (one) or greater than the cardinality of the attribute *ST_PrivatePoints*, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return an *ST_Point* value at element *aposition* in the attribute *ST_PrivatePoints* of SELF.

7.5.6 ST_StartPoint Method

Purpose

Return the start point of an ST_GeodesicString value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_GeodesicString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[1]
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.

2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[1]*.

7.5.7 ST_EndPoint Method

Purpose

Return the end point of an ST_GeodesicString value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_GeodesicString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Points()[SELF.ST_NumPoints()]
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.
2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Points()[SELF.ST_NumPoints()]*.

7.5.8 ST_GeodesicFromTxt Functions

Purpose

Return an ST_GeodesicString value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_GeodesicString value.

Definition

```
CREATE FUNCTION ST_GeodesicFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeodesicFromTxt(awkt, 0)

CREATE FUNCTION ST_GeodesicFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_GeodesicFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_GeodesicFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_GeodesicFromTxt(awkt, 0)*.
3) The function *ST_GeodesicFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeodesicFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_GeodesicString* value.

      If *awkt* is not producible in the BNF for <geodesic text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_GeodesicString)*.

7.5.9 ST_GeodesicFromWKB Functions

Purpose

Return an ST_GeodesicString value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_GeodesicString value.

Definition

```
CREATE FUNCTION ST_GeodesicFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeodesicFromWKB(awkb, 0)

CREATE FUNCTION ST_GeodesicFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_GeodesicFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_GeodesicFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_GeodesicFromWKB(awkb, 0)*.
3) The function *ST_GeodesicFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeodesicFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_GeodesicString* value.

      If *awkb* is not producible in the BNF for <geodesic binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_GeodesicString)*.

7.5.10 ST_GeodesicFromGML Functions

Purpose

Return an ST_GeodesicString value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Geodesic or GeodesicString representation of an ST_GeodesicString value.

Definition

```
CREATE FUNCTION ST_GeodesicFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeodesicFromGML(agml, 0)

CREATE FUNCTION ST_GeodesicFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_GeodesicString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_GeodesicFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_GeodesicFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_GeodesicFromGML(agml, 0)*.
3) The function *ST_GeodesicFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeodesicFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a Geodesic or GeodesicString XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_GeodesicString)*.

7.6 ST_EllipticalCurve Type and Routines

7.6.1 ST_EllipticalCurve Type

Purpose

The ST_EllipticalCurve type is a subtype of the ST_Curve type and represents a single curve segment having elliptical interpolation.

Definition

```
CREATE TYPE ST_EllipticalCurve
   UNDER ST_Curve
   AS (
      ST_PrivateReferenceLocation ST_AffinePlacement DEFAULT NULL,
      ST_PrivateUAxisLength DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateVAxisLength DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateStartAngle ST_Angle DEFAULT NULL,
      ST_PrivateEndAngle ST_Angle DEFAULT NULL,
      ST_PrivateStartM DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateEndM DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_EllipticalCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_EllipticalCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_EllipticalCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_EllipticalCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_EllipticalCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_EllipticalCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_EllipticalCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_EllipticalCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle)
   RETURNS ST_EllipticalCurve
   SELF AS RESULTk
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```



```
CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_RefLocation()
   RETURNS ST_AffinePlacement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_RefLocation
   (areflocation ST_AffinePlacement)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_UAxisLength()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_UAxisLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_UAxisLength
   (auaxislength DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_UAxisLength
   (auaxislength DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_VAxisLength()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_VAxisLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_VAxisLength
   (auaxislength DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_VAxisLength
   (auaxislength DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartAngle()
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartAngle
   (astartangle ST_Angle)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EndAngle()
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndAngle
   (anendangle ST_Angle)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

5) The attribute *ST_PrivateReferenceLocation* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferenceLocation*.

6) The attribute *ST_PrivateUAxisLength* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateUAxisLength*.

7) The attribute *ST_PrivateVAxisLength* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateVAxisLength*.

8) The attribute *ST_PrivateStartAngle* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartAngle*.

9) The attribute *ST_PrivateEndAngle* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndAngle*.

10) The attribute *ST_PrivateStartM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartM*.

11) The attribute *ST_PrivateEndM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndM*.

Description

1) The *ST_EllipticalCurve* type provides for public use:

   a) a method *ST_EllipticalCurve(CHARACTER LARGE OBJECT)*,

   b) a method *ST_EllipticalCurve(CHARACTER LARGE OBJECT, INTEGER)*,

   c) a method *ST_EllipticalCurve(BINARY LARGE OBJECT)*,

   d) a method *ST_EllipticalCurve(BINARY LARGE OBJECT, INTEGER)*,

   e) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle)*,

   f) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER)*,

   g) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION)*,

   h) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,

   i) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, CHARACTER VARYING)*,

   j) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, CHARACTER VARYING, INTEGER)*,

   k) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)*,

   l) a method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*,

   m) a method *ST_RefLocation()*,

   n) a method *ST_RefLocation(ST_AffinePlacement)*,

   o) a method *ST_UAxisLength()*,

   p) a method *ST_UAxisLength(CHARACTER VARYING)*,

   q) a method *ST_UAxisLength(DOUBLE PRECISION)*,

   r) a method *ST_UAxisLength(DOUBLE PRECISION, CHARACTER VARYING)*,

   s) a method *ST_VAxisLength()*,

   t) a method *ST_VAxisLength(CHARACTER VARYING)*,

   u) a method *ST_VAxisLength(DOUBLE PRECISION)*,

   v) a method *ST_VAxisLength(DOUBLE PRECISION, CHARACTER VARYING)*,

   w) a method *ST_StartAngle()*,

   x) a method *ST_StartAngle(ST_Angle)*,

   y) a method *ST_EndAngle()*,

   z) a method *ST_EndAngle(ST_Angle)*,

aa) a method *ST_StartM()*,

ab) a method *ST_StartM(DOUBLE PRECISION)*,

ac) a method *ST_EndM()*,

ad) a method *ST_EndM(DOUBLE PRECISION)*,

ae) an overriding method *ST_StartPoint()*,

af) an overriding method *ST_EndPoint()*,

ag) a function *ST_EllipticFromTxt(CHARACTER LARGE OBJECT)*,

ah) a function *ST_EllipticFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,

ai) a function *ST_EllipticFromWKB(BINARY LARGE OBJECT)*,

aj) a function *ST_EllipticFromWKB(BINARY LARGE OBJECT, INTEGER)*,

ak) a function *ST_EllipticFromGML(CHARACTER LARGE OBJECT)*,

al) a function *ST_EllipticFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateReferenceLocation* attribute contains the *ST_AffinePlacement* reference location value.

3) The *ST_PrivateUAxisLength* attribute contains the DOUBLE PRECISION u axis length value.

4) The *ST_PrivateVAxisLength* attribute contains the DOUBLE PRECISION v axis length value.

5) The *ST_PrivateStartAngle* attribute contains the *ST_Angle* start angle value.

6) The *ST_PrivateEndAngle* attribute contains the *ST_Angle* end angle value.

7) The *ST_PrivateStartM* attribute contains the DOUBLE PRECISION start measure value.

8) The *ST_PrivateEndM* attribute contains the DOUBLE PRECISION end measure value.

9) An *ST_EllipticalCurve* is not well formed if *ST_PrivateStartM* is NULL and *ST_PrivateEndM* is NOT NULL or if *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NULL.

10) If *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NOT NULL, then *SELF.ST_IsMeasured* equals 1 (one). Otherwise, *SELF.ST_IsMeasured* equals 0 (zero).

11) The *ST_EllipticalCurve ST_PrivateIs3D* attribute value shall be equal to the *ST_PrivateIs3D* attribute value of the *ST_PrivateReferenceLocation ST_AffinePlacement ST_Point ST_PrivateLocation* attribute value.

12) Let *AV* be the *ST_PrivateReferenceDirections* attribute *ST_Vector* ARRAY value of the *ST_EllipticalCurve ST_PrivateReferenceLocation* attribute *ST_AffinePlacement* value. Then the *ST_EllipticalCurve ST_PrivateIs3D* value shall be equal to one (1) if and only if the value returned by the *ST_GetCoordDim(AV)* function is equal to 3.

13) The cardinality of the *ST_PrivateReferenceLocation ST_AffinePlacement ST_Vector* ARRAY *ST_PrivateReferenceDirections* attribute value shall be equal to 2.

14) The type *ST_EllipticalCurve* is a subtype of *ST_Curve* with elliptical interpolation.

15) The type *ST_EllipticalCurve* defines a single elliptical curve segment.

16) An *ST_EllipticalCurve* having start and end angles that differ by 360 degrees is a complete *ellipse*. It is closed and simple and is therefore a *ring*.

17) An *ST_EllipticalCurve* value returned by the constructor function corresponds to the empty set.

18) For any *ST_Angle* value, the (local) coordinates of a point on the *ST_EllipticalCurve* "at that angle in the 2D parameter space" are

    (u,v) = (uAxisLength * cos (angle) , vAxisLength * sin (angle))

19) The start and endpoint coordinates are obtained with the *ST_StartAngle* and *ST_EndAngle*, respectively

    (uStart,vStart) = (uAxisLength * cos(startAngle), vAxisLength * sin(startAngle))

(uEnd,vEnd) = (uAxisLength * cos(endAngle), vAxisLength * sin(endAngle))

20) If *SELF.ST_IsMeasured* equals 1 (one), then the *ST_PrivateStartM* and ST*_PrivateEndM* m coordinate values shall be added to the start and endpoint coordinates.

21) The contained portion of the (partial) elliptical curve consists of all angles "between" the start angle and end angle in the simple numerical way. If the start angle is less than the end angle, the curve is swept in a counterclockwise direction from the start angle up to the end angle. If the start angle is greater than the end angle, the curve is swept in a clockwise direction from the start angle down to the end angle. Reversing the order of start angle and end angle traces the same curve in the opposite direction. Either or both angles may be negative or larger than 360 to obtain proper control of arcs that wrap around the positive or negative u axis. For example, the four possible cases with 0 and 90 degrees are as follows:

    start = 0, end = 90: sweeps counterclockwise through 90 degrees from 0 up to 90,

    start = 90, end = 0: sweeps clockwise through 90 degrees from 90 down to 0,

    start = 90, end = 360: sweeps counterclockwise through 270 degrees from 90 up to 360,

    start = 360, end = 90: sweeps clockwise through 270 degrees from 360 down to 90.

7.6.2 ST_EllipticalCurve Methods

Purpose

Return an ST_EllipticalCurve value constructed from either the well-known text representation; the well-known binary representation; the GML representation; or the specified reference location ST_AffinePlacement value, DOUBLE PRECISION uAxisLength and vAxisLength values, and the start and end ST_Angle values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN ST_EllipticalFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(areferencelocation, auaxislength,
      avaxislength, astartangle, anendangle, NULL, NULL, 0)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(areferencelocation, auaxislength,
      avaxislength, astartangle, anendangle, NULL, NULL, ansrid)
```

```
CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(areferencelocation, auaxislength,
      avaxislength, astartangle, anendangle, astartm, anendm, 0)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(areferencelocation, auaxislength,
     avaxislength, astartangle, anendangle, NULL, NULL, aunit, 0)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(areferencelocation, auaxislength,
      avaxislength, astartangle, anendangle, NULL, NULL, aunit, ansrid)
```

```
CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   RETURN NEW ST_EllipticalCurve(areferencelocation, auaxislength,
      avaxislength, astartangle, anendangle, astartm, anendm, aunit, 0)

CREATE CONSTRUCTOR METHOD ST_EllipticalCurve
   (areferencelocation ST_AffinePlacement,
    auaxislength DOUBLE PRECISION,
    avaxislength DOUBLE PRECISION,
    astartangle ST_Angle,
    anendangle ST_Angle,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Geometry* value.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_EllipticalCurve(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_EllipticalCurve(awktorgml, 0)*.
3) The method *ST_EllipticalCurve(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

a) If *awktorgml* contains an EllipticalCurve XML element in the GML representation, then return the result of the value expression: *ST_EllipticFromGML(awktorgml, ansrid)*.

b) If *awktorgml* contains an Ellipse XML element in the GML representation, then return the result of the value expression: *ST_EllipticFromGML(awktorgml, ansrid)*.

c) Otherwise, return the result of the value expression: *ST_EllipticFromTxt(awktorgml, ansrid)*.

5) The method *ST_EllipticalCurve(BINARY LARGE OBJECT)* takes the following input parameter:

a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_EllipticalCurve(awkb, 0)*.

7) The method *ST_EllipticalCurve(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a BINARY LARGE OBJECT value *awkb*,

b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_EllipticFromWKB(awkb, ansrid)*.

9) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *auaxislength,*

c) a DOUBLE PRECISION value *avaxislength,*

d) an *ST_Angle* value *astartangle,*

e) an *ST_Angle* value *anendangle*.

10) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle)* returns the result of the value expression: *NEW ST_EllipticalCurve(areferencelocation, auaxislength, avaxislength, astartangle, anendangle, NULL, NULL, 0)*.

11) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *auaxislength,*

c) a DOUBLE PRECISION value *avaxislength,*

d) an *ST_Angle* value *astartangle,*

e) an *ST_Angle* value *anendangle*,

f) an INTEGER value *ansrid.*

12) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER)* returns the result of the value expression: *NEW ST_EllipticalCurve(areferencelocation, auaxislength, avaxislength, astartangle, anendangle, NULL, NULL, ansrid)*.

13) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *auaxislength*,

c) a DOUBLE PRECISION value *avaxislength*,

d) an *ST_Angle* value *astartangle*,

e) an *ST_Angle* value *anendangle*,

f) a DOUBLE PRECISION value *astartm*,

g) a DOUBLE PRECISION value *anendm*.

14) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_EllipticalCurve(areferencelocation, auaxislength, avaxislength, astartangle, anendangle, astartm, anendm, 0)*.

15) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *auaxislength*,

c) a DOUBLE PRECISION value *avaxislength*,

d) an *ST_Angle* value *astartangle*,

e) an *ST_Angle* value *anendangle*,

f) a DOUBLE PRECISION value *astartm*,

g) a DOUBLE PRECISION value *anendm*,

h) an INTEGER value *ansrid*.

16) Case:

a) If the spatial reference system ansrid defines a <linear unit>, then the values *auaxislength* and *avaxislength* are in the linear unit of measure identified by <linear unit>.

b) Otherwise, the values *auaxislength* and *avaxislength* are in an implementation-defined unit of measure.

17) For the type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

Case:

a) If *areferencelocation* is the null value or if *auaxislength* is the null value or if *avaxislength* is the null value or if *astartangle* is the null value or if *anendangle* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_EllipticalCurve* value with:

i) The *ST_PrivateDimension* attribute set to 1 (one).

ii) Case:

1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

iii) The *ST_PrivateCoordinateDimension* attribute set to *areferencelocation.ST_Location.ST_CoordDim + ST_PrivateIsMeasured*.

iv) The *ST_PrivateIs3D* attribute set to *areferencelocation.ST_Location.ST_Is3D()*.

v) Case:

1) If CARDINALITY(*areferencelocation.ST_RefDirections()*) not equal to 2, then an exception is raised: *SQL/MM Spatial exception – incorrect number of vectors*.

2) Otherwise, the *ST_PrivateReferenceLocation* attribute set to *areferencelocation*.

vi) The *ST_PrivateUAxisLength* attribute set to *auaxislength*.

vii) The *ST_PrivateVAxisLength* attribute set to *avaxislength*.

viii) The *ST_PrivateStartAngle* attribute set to *astartangle*.

ix) The *ST_PrivateEndAngle* attribute set to *anendangle*.

x) The *ST_PrivateStartM* attribute set to *astartm*.

xi) The *ST_PrivateEndM* attribute set to *anendm*.

xii) The spatial reference system identifier set to *ansrid*.

18) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *auaxislength,*

c) a DOUBLE PRECISION value *avaxislength,*

d) an *ST_Angle* value *astartangle,*

e) an *ST_Angle* value *anendangle*,

f) a CHARACTER VARYING value *aunit.*

19) The null-call type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_EllipticalCurve(areferencelocation, auaxislength, avaxislength, astartangle, anendangle, NULL, NULL, aunit, 0)*.

20) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, CHARACTER VARYING, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *auaxislength,*

c) a DOUBLE PRECISION value *avaxislength,*

d) an *ST_Angle* value *astartangle,*

e) an *ST_Angle* value *anendangle*,

f) a CHARACTER VARYING value *aunit,*

g) an INTEGER value *ansrid.*

21) The null-call type-preserving SQL-invoked constructor method ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, INTEGER) returns the result of the value expression: *NEW ST_EllipticalCurve(areferencelocation, auaxislength, avaxislength, astartangle, anendangle, NULL, NULL, aunit, ansrid)*.

22) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *auaxislength,*

c) a DOUBLE PRECISION value *avaxislength,*

d) an *ST_Angle* value *astartangle*,

e) an *ST_Angle* value *anendangle*,

f) a DOUBLE PRECISION value *astartm,*

g) a DOUBLE PRECISION value *anendm*,

h) a CHARACTER VARYING value *aunit.*

23) The type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_EllipticalCurve(areferencelocation, auaxislength, avaxislength, astartangle, anendangle, astartm, anendm, aunit, 0)*.

24) The method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *auaxislength,*

c) a DOUBLE PRECISION value *avaxislength,*

d) an *ST_Angle* value *astartangle*,

e) an *ST_Angle* value *anendangle*,

f) a DOUBLE PRECISION value *astartm,*

g) a DOUBLE PRECISION value *anendm*,

h) a CHARACTER VARYING value *aunit,*

i) an INTEGER value *ansrid.*

25) For the values *auaxislength* and *avaxislength*:

a) The value for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

26) For the type-preserving SQL-invoked constructor method *ST_EllipticalCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, ST_Angle, ST_Angle, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*:

Case:

a) If *areferencelocation* is the null value or if *auaxislength* is the null value or if *avaxislength* is the null value or if *astartangle* is the null value or if *anendangle* is the null value or if *aunit* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_EllipticalCurve* value with:

i) The *ST_PrivateDimension* attribute set to 1 (one).

ii) Case:

1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

iii) The *ST_PrivateCoordinateDimension* attribute set to *areferencelocation.ST_Location.ST_CoordDim* + *ST_PrivateIsMeasured*.

iv) The *ST_PrivateIs3D* attribute set to *areferencelocation.ST_Location.ST_Is3D*.

v) Case:

1) If CARDINALITY(areferencelocation.ST_RefDirections()) not equal to 2, then an exception is raised: SQL/MM Spatial exception – incorrect number of vectors.

2) Otherwise, the ST_PrivateReferenceLocation attribute set to areferencelocation.

vi) The *ST_PrivateUAxisLength* attribute set to *auaxislength*.

vii) The *ST_PrivateVAxisLength* attribute set to *avaxislength*.

viii) The ST_PrivateStartAngle attribute set to astartangle.

ix) The *ST_PrivateEndAngle* attribute set to *anendangle*.

x) The *ST_PrivateStartM* attribute set to *astartm*.

xi) The *ST_PrivateEndM* attribute set to *anendm*.

xii) The spatial reference system identifier set to *ansrid*.

7.6.3 ST_RefLocation Methods

Purpose

Observe and mutate the attribute ST_PrivateReferenceLocation of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_RefLocation()
   RETURNS ST_AffinePlacement
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateReferenceLocation
      END

CREATE METHOD ST_RefLocation
   (areflocation ST_AffinePlacement)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If areflocation is the null value, then raise an exception
      IF areflocation IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      -- Check if curve and affine placement location point are both 2D
      -- or both 3D
      IF SELF.ST_Is3D() <> areflocation.ST_Location().ST_Is3D() THEN
         SIGNAL SQLSTATE '2FF96'
            SET MESSAGE_TEXT = 'mixed Is3D';
      END IF;
      -- Check if affine placement reference directions has two vectors
      IF CARDINALITY(areflocation.ST_RefDirections()) <> 2 THEN
         SIGNAL SQLSTATE '2FF86'
            SET MESSAGE_TEXT = 'incorrect number of vectors';
      END IF;
      RETURN
         SELF.ST_PrivateReferenceLocation(areflocation);
   END
```

Description

1) The method *ST_RefLocation()* has no input parameters.

2) For the null-call method *ST_RefLocation()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateReferenceLocation* of SELF.

3) The method *ST_RefLocation(ST_AffinePlacement)* takes the following input parameters:

   a) an *ST_AffinePlacement* value *areflocation*.

4) For the type-preserving method *ST_RefLocation(ST_AffinePlacement)*:

   Case:

a) If *areflocation* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

. c) If SELF and the *ST_PrivateLocation* attribute *ST_Point* value of *areflocation* are not either both 2D or both 3D, then an exception condition is raised: *SQL/MM Spatial exception – mixed Is3D*.

d) If the number of *ST_Vectors* in the *ST_PrivateReferenceDirections* attribute of *areflocation* is not equal to 2, then an exception condition is raised: *SQL/MM Spatial exception – incorrect number of vectors.*

e) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateReferenceLocation* set to *areflocation*.

7.6.4 ST_UAxisLength Methods

Purpose

Observe and mutate the attribute ST_PrivateUAxisLength of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_UAxisLength()
   RETURNS DOUBLE PRECISION
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateUAxisLength
      END

CREATE METHOD ST_UAxisLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateUAxisLength
      END

CREATE METHOD ST_UAxisLength
   (auaxislength DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If auaxislength is the null value, then raise an exception
      IF auaxislength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      RETURN
         SELF.ST_PrivateUAxisLength(auaxislength);
   END
```

```
CREATE METHOD ST_UAxisLength
   (auaxislength DOUBLE PRECISION,
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If auaxislength is the null value, then raise an exception
      IF auaxislength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      RETURN
         SELF.ST_PrivateUAxisLength(auaxislength);
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_UAxisLength()* has no input parameters.
2) For the null-call method *ST_UAxisLength()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateUAxisLength* of SELF.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_UAxisLength()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_UAxisLength()* is in an implementation-defined unit of measure.
3) The method *ST_UAxisLength(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_UAxisLength(CHARACTER VARYING)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateUAxisLength* of SELF.
5) For the method *ST_UAxisLength(CHARACTER VARYING)*:
   a) The value for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
6) The method *ST_UAxisLength(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *auaxislength*.

7) For the type-preserving method *ST_UAxisLength(DOUBLE PRECISION)*:

   Case:

   a) If *auaxislength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateUAxisLength* set to *auaxislength*.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_UAxisLength()* is in the linear unit of measure identified by <linear unit>.

      ii) Otherwise, the value returned by *ST_UAxisLength()* is in an implementation-defined unit of measure.

8) The method *ST_UAxisLength(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

   a) a DOUBLE PRECISION value *auaxislength*,

   b) a CHARACTER VARYING value *aunit*.

9) For the type-preserving method *ST_UAxisLength(DOUBLE PRECISION, CHARACTER VARYING)*:

   Case:

   a) If *auaxislength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateUAxisLength* set to *auaxislength*.

10) For the method *ST_UAxisLength(DOUBLE PRECISION, CHARACTER VARYING)*:

   a) The value for *aunit* shall be a supported <unit name>.

   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

7.6.5 ST_VAxisLength Methods

Purpose

Observe and mutate the attribute ST_PrivateVAxisLength of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_VAxisLength()
   RETURNS DOUBLE PRECISION
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateVAxisLength
      END

CREATE METHOD ST_VAxisLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateVAxisLength
      END

CREATE METHOD ST_VAxisLength
   (avaxislength DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If avaxislength is the null value, then raise an exception
      IF avaxislength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      RETURN
         SELF.ST_PrivateVAxisLength(avaxislength);
   END
```

```
CREATE METHOD ST_VAxisLength
   (avaxislength DOUBLE PRECISION,
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If avaxislength is the null value, then raise an exception
      IF avaxislength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      RETURN
         SELF.ST_PrivateVAxisLength(avaxislength);
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_VAxisLength()* has no input parameters.
2) For the null-call method *ST_VAxisLength()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateVAxisLength* of SELF.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_VAxisLength()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_VAxisLength()* is in an implementation-defined unit of measure.
3) The method *ST_VAxisLength(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_VAxisLength(CHARACTER VARYING)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateVAxisLength* of SELF.
5) For the method *ST_VAxisLength(CHARACTER VARYING)*:
   a) The value for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
6) The method *ST_VAxisLength(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *avaxislength*.

7) For the type-preserving method *ST_VAxisLength(DOUBLE PRECISION)*:

   Case:

   a) If *avaxislength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateVAxisLength* set to *avaxislength*.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_VAxisLength()* is in the linear unit of measure identified by <linear unit>.

      ii) Otherwise, the value returned by *ST_VAxisLength()* is in an implementation-defined unit of measure.

8) The method *ST_VAxisLength(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

   a) a DOUBLE PRECISION value *avaxislength*,

   b) a CHARACTER VARYING value *aunit*.

9) For the type-preserving method *ST_VAxisLength(DOUBLE PRECISION, CHARACTER VARYING)*:

   Case:

   a) If *avaxislength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateVAxisLength* set to *avaxislength*.

10) For the method *ST_VAxisLength(DOUBLE PRECISION, CHARACTER VARYING)*:

   a) The value for *aunit* shall be a supported <unit name>.

   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

7.6.6 ST_StartAngle Methods

Purpose

Observe and mutate the attribute ST_PrivateStartAngle of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_StartAngle()
   RETURNS ST_Angle
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartAngle
      END

CREATE METHOD ST_StartAngle
   (astartangle ST_Angle)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If astartangle is the null value, then raise an exception
      IF astartangle IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      RETURN
         SELF.ST_PrivateStartAngle(astartangle);
   END
```

Description

1) The method *ST_StartAngle()* has no input parameters.
2) For the null-call method *ST_StartAngle()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateStartAngle* of SELF.
3) The method *ST_StartAngle(ST_Angle)* takes the following input parameters:
   a) an *ST_Angle* value *astartangle*.
4) For the type-preserving method *ST_StartAngle(ST_Angle)*:

   Case:

   a) If *astartangle* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateStartAngle* set to *astartangle*.

7.6.7 ST_EndAngle Methods

Purpose

Observe and mutate the attribute ST_PrivateEndAngle of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_EndAngle()
   RETURNS ST_Angle
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndAngle
      END

CREATE METHOD ST_EndAngle
   (anendangle ST_Angle)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      -- If anendangle is the null value, then raise an exception
      IF anendangle IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      RETURN
         SELF.ST_PrivateEndAngle(anendangle);
   END
```

Description

1) The method *ST_EndAngle()* has no input parameters.

2) For the null-call method *ST_EndAngle()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateEndAngle* of SELF.

3) The method *ST_EndAngle(ST_Angle)* takes the following input parameters:

   a) an *ST_Angle* value *anendangle*.

4) For the type-preserving method *ST_EndAngle(ST_Angle)*:

   Case:

   a) If *anendangle* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_EllipticalCurve* value with the attribute *ST_PrivateEndAngle* set to *anendangle*.

7.6.8 ST_StartM Methods

Purpose

Observe and mutate the attribute ST_PrivateStartM of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartM
      END

CREATE METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateEndM is NOT NULL THEN
         SET measured = 1;
      -- If astartm is NULL, IS_Measured must be 0 (zero)
      IF astartm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateStartM(astartm);
   END
```

Description

1) The method *ST_StartM()* has no input parameters.

2) For the null-call method *ST_StartM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateStartM* of SELF.

3) The method *ST_StartM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *astartm*.

4) For the type-preserving method *ST_StartM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0 (zero).

      ii) If *SELF.ST_PrivateEndM* is NOT NULL, set *M* = 1 (one).

      iii) If *astartm* IS NULL, set *M* to 0 (zero).

iv) Return an *ST_EllipticalCurve* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_StartM* attribute set to *astartm*.

7.6.9 ST_EndM Methods

Purpose

Observe and mutate the attribute ST_PrivateEndM of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndM
      END

CREATE METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_EllipticalCurve
   FOR ST_EllipticalCurve
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_EllipticalCurve);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateStartM is NOT NULL THEN
         SET measured = 1;
      -- If anendm is NULL, IS_Measured must be 0 (zero)
      IF anendm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateEndM(anendm);
   END
```

Description

1) The method *ST_EndM()* has no input parameters.

2) For the null-call method *ST_EndM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateEndM* of SELF.

3) The method *ST_EndM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *anendm*.

4) For the type-preserving method *ST_EndM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* 0 (zero).

      ii) If *SELF.ST_PrivateStartM* is NOT NULL, set *M* = 1 (one).

      iii) If *anendm* IS NULL, set *M* to 0 (zero).

iv) Return an *ST_EllipticalCurve* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_EndM* attribute set to *anendm*.

7.6.10 ST_StartPoint Method

Purpose

Return the start point of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.

2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *SP* be the *ST_Point* start point value calculated from the *ST_PrivateReferenceLocation, ST_PrivateUAxisLength, ST_PrivateVAxisLength,* and *ST_PrivateStartAngle* attribute values of SELF.

      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *SP = SP.ST_M(SELF.ST_StartM())*.

      iii) Return *SP*.

7.6.11 ST_EndPoint Method

Purpose

Return the end point of an ST_EllipticalCurve value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_EllipticalCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.

2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *EP* be the *ST_Point* end point value calculated from the *ST_PrivateReferenceLocation, ST_PrivateUAxisLength, ST_PrivateVAxisLength,* and *ST_PrivateEndAngle* attribute values of SELF.

      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *EP = EP.ST_M(SELF.ST_EndM())*.

      iii) Return *EP*.

7.6.12 ST_EllipticFromTxt Functions

Purpose

Return an ST_EllipticalCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_EllipticalCurve value.

Definition

```
CREATE FUNCTION ST_EllipticFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_EllipticFromTxt(awkt, 0)

CREATE FUNCTION ST_EllipticFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_EllipticFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_EllipticFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_EllipticFromTxt(awkt, 0)*.
3) The function *ST_EllipticFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_EllipticFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_EllipticalCurve* value.

      If *awkt* is not producible in the BNF for <elliptical text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_EllipticalCurve)*.

7.6.13 ST_EllipticFromWKB Functions

Purpose

Return an ST_EllipticalCurve value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_EllipticalCurve value.

Definition

```
CREATE FUNCTION ST_EllipticFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_EllipticFromWKB(awkb, 0)

CREATE FUNCTION ST_EllipticFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_EllipticFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_EllipticFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_EllipticFromWKB(awkb, 0)*.
3) The function *ST_EllipticFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_EllipticFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkb* is the well-known binary representation of an *ST_EllipticalCurve* value.

      If *awkb* is not producible in the BNF for <elliptical binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_EllipticalCurve)*.

7.6.14 ST_EllipticFromGML Functions

Purpose

Return an ST_EllipticalCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML EllipticalCurve or Ellipse representation of an ST_EllipticalCurve value.

Definition

```
CREATE FUNCTION ST_EllipticFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_EllipticFromGML(agml, 0)

CREATE FUNCTION ST_EllipticFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_EllipticalCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_EllipticFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_EllipticFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_EllipticFromGML(agml, 0)*.
3) The function *ST_EllipticFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_EllipticFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain an EllipticalCurve or Ellipse XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_EllipticalCurve)*.

7.7 ST_NURBSCurve Type and Routines

7.7.1 ST_NURBSCurve Type

Purpose

The ST_NURBSCurve type is a subtype of the ST_Curve type and represents a single, continuous curve segment Non-Uniform Rational BSpline curve.

Definition

```
CREATE TYPE ST_NURBSCurve
   UNDER ST_Curve
   AS (
      ST_PrivateDegree INTEGER DEFAULT NULL,
      ST_PrivateControlPoints ST_NURBSPoint
         ARRAY[ST_MaxNURBSPointArrayElements] DEFAULT ARRAY[],
      ST_PrivateKnots ST_Knot ARRAY[ST_MaxKnotArrayElements]
         DEFAULT ARRAY[],
      ST_PrivateStartM DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateEndM DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_NURBSCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_NURBSCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_NURBSCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_NURBSCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_NURBSCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_NURBSCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_NURBSCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_NURBSCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_NURBSCurve
   (degree INTEGER,
    controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements],
    knots ST_Knot ARRAY[ST_MaxKnotArrayElements])
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_NURBSCurve
   (degree INTEGER,
    controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements],
    knots ST_Knot ARRAY[ST_MaxKnotArrayElements],
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_NURBSCurve
   (degree INTEGER,
    controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements],
    knots ST_Knot ARRAY[ST_MaxKnotArrayElements],
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_NURBSCurve
   (degree INTEGER,
    controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements],
    knots ST_Knot ARRAY[ST_MaxKnotArrayElements],
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Degree()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_ControlPoints()
   RETURNS ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ControlPoints
   (controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements])
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Knots()
   RETURNS ST_Knot ARRAY[ST_MaxKnotArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Knots
   (knots ST_Knot ARRAY[ST_MaxKnotArrayElements])
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_NURBSCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxNURBSPointArrayElements* is the implementation-defined maximum cardinality of an array of *ST_NURBSPoint* values.

2) *ST_MaxKnotArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Knot* values.

3) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

4) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

5) The attribute *ST_PrivateDegree* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateDegree*.

6) The attribute *ST_PrivateControlPoints* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateControlPoints*.

7) The attribute *ST_PrivateKnots* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateKnots*.

8) The attribute *ST_PrivateStartM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartM*.

9) The attribute *ST_PrivateEndM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndM*.

Description

1) The *ST_NURBSCurve* type provides for public use:

   a) a method *ST_NURBSCurve(CHARACTER LARGE OBJECT)*,

   b) a method *ST_NURBSCurve(CHARACTER LARGE OBJECT, INTEGER)*,

   c) a method *ST_NURBSCurve(BINARY LARGE OBJECT)*,

   d) a method *ST_NURBSCurve(BINARY LARGE OBJECT, INTEGER)*,

   e) a method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY)*,

   f) a method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, INTEGER)*,

   g) a method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION)*,

   h) a method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,

   i) a method *ST_Degree()*,

   j) a method *ST_ControlPoints()*,

   k) a method *ST_ControlPoints(ST_NURBSPoint ARRAY)*,

l) a method *ST_Knots()*,

m) a method *ST_Knots(ST_Knot ARRAY)*,

n) a method *ST_StartM()*,

o) a method *ST_StartM(DOUBLE PRECISION)*,

p) a method *ST_EndM()*,

q) a method *ST_EndM(DOUBLE PRECISION)*,

r) an overriding method *ST_StartPoint()*,

s) an overriding method *ST_EndPoint()*,

t) a function *ST_NURBSFromTxt(CHARACTER LARGE OBJECT)*,

u) a function *ST_NURBSFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,

v) a function *ST_NURBSFromWKB(BINARY LARGE OBJECT)*,

w) a function *ST_NURBSFromWKB(BINARY LARGE OBJECT, INTEGER)*,

x) a function *ST_NURBSFromGML(CHARACTER LARGE OBJECT)*,

y) a function *ST_NURBSFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateDegree* attribute contains the INTEGER degree value.

3) The *ST_PrivateControlPoints* attribute contains the *ST_NURBSPoint* ARRAY control points value.

4) The *ST_PrivateKnots* attribute contains the *ST_Knot* ARRAY knots value.

5) The *ST_PrivateStartM* attribute contains the DOUBLE PRECISION start measure value.

6) The *ST_PrivateEndM* attribute contains the DOUBLE PRECISION end measure value.

7) An *ST_NURBSCurve* is not well formed if *ST_PrivateStartM* is NULL and *ST_PrivateEndM* is NOT NULL or if *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NULL.

8) If *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NOT NULL, then *SELF.ST_IsMeasured* equals 1 (one). Otherwise, *SELF.ST_IsMeasured* equals 0 (zero).

9) The coordinate dimension of an *ST_NURBSCurve* value shall be equal to the coordinate dimension of the *ST_NURBSPoint* ARRAY *ST_NURBSPoint ST_Point ST_PrivateWeightedPoint* attribute values.

10) If *SELF.ST_IsMeasured* equals 1 (one) the coordinate dimension of the *ST_NURBSCurve* value shall be increased by 1 (one).

11) The type *ST_NURBSCurve* is a subtype of *ST_Curve*.

12) The type *ST_NURBSCurve* defines a single NURBS curve segment.

13) If *SELF.ST_IsMeasured* equals 1 (one), then the *ST_PrivateStartM* and ST_*PrivateEndM* m coordinate values shall be included in the start and endpoint coordinates for *ST_StartPoint* and *ST_EndPoint*, respectively.

14) An *ST_NURBSCurve* having an *ST_StartPoint* value equal to its *ST_EndPoint* value is closed and simple and is therefore a *ring*.

15) An *ST_NURBSCurve* value returned by the constructor function corresponds to the empty set.

16) The total number of knots (number of distinct knot values times their respective multipliers) must be equal to the number of control points plus the degree plus 1 (one).

17) If weights are present, the control points are "weighted" – i.e. the composite weighted control point (wx,wy,wz,w) has its wx,wy,wz values stored in the control points value, where wx, for example, is usually the weight times the cartesian x coordinate value (not the cartesian x coordinate value).

18) In conventional usage, weights are always positive.

19) The knots are NOT required to be "clamped". (A "clamped" knot array has {degree+1} identical values of the minimum knot and {degree+1} identical values of the maximum knot. This causes the spline curve to pass exactly through the first and final control points.

20) There is no separate "periodic" curve. Periodic curves are to be produced by having the first and last {degree+1} knots differ by precisely the knot space period.

21) An *ST_NURBSCurve* value shall not have knot multiplicities greater than the degree of the curve, as this would result in a gap in the curve.

7.7.2 ST_NURBSCurve Methods

Purpose

Return an ST_NURBSCurve value constructed from either the well-known text representation; the well-known binary representation; the GML representation; or the specified INTEGER degree, ST_NURBSPoint ARRAY control points, and the ST_Knot ARRAY knots values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_NURBSCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   RETURN NEW ST_NURBSCurve(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_NURBSCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_NURBSCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   RETURN NEW ST_NURBSCurve(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_NURBSCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   RETURN ST_NURBSFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_NURBSCurve
      (degree INTEGER,
       controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements],
       knots ST_Knot ARRAY[ST_MaxKnotArrayElements])
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   RETURN NEW ST_NURBSCurve(degree, controlpoints,
     knots, NULL, NULL, 0)

CREATE CONSTRUCTOR METHOD ST_NURBSCurve
      (degree INTEGER,
       controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements],
       knots ST_Knot ARRAY[ST_MaxKnotArrayElements],
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   RETURN NEW ST_NURBSCurve(degree, controlpoints,
      knots, NULL, NULL, ansrid)
```

```
CREATE CONSTRUCTOR METHOD ST_NURBSCurve
   (degree INTEGER,
    controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements]],
    knots ST_Knot ARRAY[ST_MaxKnotArrayElements])
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   RETURN NEW ST_NURBSCurve(degree, controlpoints,
      knots, astartm, anendm, 0)

CREATE CONSTRUCTOR METHOD ST_NURBSCurve
   (degree INTEGER,
    controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements]],
    knots ST_Knot ARRAY[ST_MaxKnotArrayElements],
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNURBSPointArrayElements* is the implementation-defined maximum cardinality of an array of *ST_NURBSPoint* values.

2) *ST_MaxKnotArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Knot* values.

3) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

4) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Geometry* value.

Description

1) The method *ST_NURBSCurve(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_NURBSCurve(awktorgml, 0)*.

3) The method *ST_NURBSCurve(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a BSpline XML element in the GML representation, then return the result of the value expression: *ST_NURBSFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_NURBSFromTxt(awktorgml, ansrid)*.

5) The method *ST_NURBSCurve(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_NURBSCurve(awkb, 0)*.

7) The method *ST_NURBSCurve(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_NURBSFromWKB(awkb, ansrid)*.

9) The method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY)* takes the following input parameters:

   a) an INTEGER value *degree,*

   b) an *ST_NURBSPoint* ARRAY value *controlpoints,*

   c) an *ST_Knot* ARRAY value *knots.*

10) The null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY)* returns the result of the value expression: *NEW ST_NURBSCurve(degree, controlpoints, knots, NULL, NULL, 0)*.

11) The method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, INTEGER)* takes the following input parameters:

   a) an INTEGER value *degree,*

   b) an *ST_NURBSPoint* ARRAY value *controlpoints,*

   c) an *ST_Knot* ARRAY value *knots,*

   d) an INTEGER value *ansrid.*

12) The null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, INTEGER)* returns the result of the value expression: *NEW ST_NURBSCurve(degree, controlpoints, knots, NULL, NULL, ansrid)*.

13) The method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

   a) an INTEGER value *degree,*

   b) an *ST_NURBSPoint* ARRAY value *controlpoints*,

   c) an *ST_Knot* ARRAY value *knots*,

   d) a DOUBLE PRECISION value *astartm,*

   e) a DOUBLE PRECISION value *anendm*.

14) The null-call type-preserving SQL-invoked constructor method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_NURBSCurve(degree, controlpoints, knots, astartm, anendm, 0)*.

15) The method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) an INTEGER value *degree,*

   b) an *ST_NURBSPoint* ARRAY value *controlpoints*,

   c) an *ST_Knot* ARRAY value *knots,*

   d) a DOUBLE PRECISION value *astartm,*

   e) a DOUBLE PRECISION value *anendm,*

   f) an INTEGER value *ansrid.*

16) For the type-preserving SQL-invoked constructor method *ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

    Case:

    a) If *degree* is the null value or if *controlpoints* is the null value or if *knots* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

    b) If SELF is the null value, then return the null value.

    c) Otherwise, return an *ST_NURBSCurve* value with:

        i) The *ST_PrivateDimension* attribute set to 1 (one).

        ii) Case:

            1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

            2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

            3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

        iii) The *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(controlpoints)* + *ST_PrivateIsMeasured*.

        iv) The *ST_PrivateIs3D* attribute set to *ST_GetIs3D(controlpoints).*

        v) The ST_PrivateDegree attribute set to *degree*.

        vi) The *ST_PrivateControlPoints* attribute set to *controlpoints*.

        vii) The *ST_PrivateKnots* attribute set to *knots*.

        viii) The *ST_PrivateStartM* attribute set to *astartm*.

        ix) The *ST_PrivateEndM* attribute set to *anendm*.

        x) The spatial reference system identifier set to *ansrid*.

7.7.3 ST_Degree Method

Purpose

Return the attribute ST_PrivateDegree of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_Degree()
   RETURNS INTEGER
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateDegree
      END
```

Description

1) The method *ST_Degree()* has no input parameters.
2) For the null-call method *ST_Degree()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateDegree* of SELF.

7.7.4 ST_ControlPoints Methods

Purpose

Observe and mutate the attribute ST_PrivateControlPoints of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_ControlPoints()
   RETURNS ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements]
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateControlPoints
      END

CREATE METHOD ST_ControlPoints
   (controlpoints ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements])
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   BEGIN
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_NURBSCurve);
      END IF;
      RETURN
         SELF.ST_PrivateControlPoints(controlpoints);
   END
```

Definitional Rules

1) *ST_MaxNURBSPointArrayElements* is the implementation-defined maximum cardinality of an array of *ST_NURBSPoint* values.

Description

1) The method *ST_ControlPoints()* has no input parameters.
2) For the null-call method *ST_ControlPoints()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateControlPoints* of SELF.
3) The method *ST_ControlPoints(ST_NURBSPoint ARRAY)* takes the following input parameters:
   a) an *ST_NURBSPoint* ARRAY value *controlpoints*.
4) For the type-preserving method *ST_ControlPoints(ST_NURBSPoint ARRAY)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_NURBSCurve* value with:
      i) The *ST_PrivateControlPoints* attribute set to *controlpoints*.
      ii) The *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(controlpoints).*
      iii) The *ST_PrivateIs3D* attribute set to:

         Case:

         1) If *ST_GetCoordDim(controlpoints)* = 3, then 1 (one).

2) Otherwise 0 (zero).

iv) The *ST_PrivateIsMeasured* attribute set to 0 (zero).

7.7.5 ST_Knots Methods

Purpose

Observe and mutate the attribute ST_PrivateKnots of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_Knots()
   RETURNS ST_Knot ARRAY[ST_MaxKnotArrayElements]
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateKnots
      END

CREATE METHOD ST_Knots
   (knots ST_Knot ARRAY[ST_MaxKnotArrayElements])
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   BEGIN
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_NURBSCurve);
      END IF;
      RETURN
         SELF.ST_PrivateKnots(knots);
   END
```

Definitional Rules

1) *ST_MaxKnotArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Knot* values.

Description

1) The method *ST_Knots()* has no input parameters.
2) For the null-call method *ST_Knots()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateKnots* of SELF.
3) The method *ST_Knots(ST_Knot ARRAY)* takes the following input parameters:
   a) an *ST_Knot* ARRAY value *knots*.
4) For the type-preserving method *ST_Knots(ST_Knot ARRAY)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_NURBSCurve* value with the attribute *ST_PrivateKnots* set to *knots*.

7.7.6 ST_StartM Methods

Purpose

Observe and mutate the attribute ST_PrivateStartM of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartM
      END

CREATE METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_NURBSCurve);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateEndM is NOT NULL THEN
         SET measured = 1;
      -- If astartm is NULL, IS_Measured must be 0 (zero)
      IF astartm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateStartM(astartm);
   END
```

Description

1) The method *ST_StartM()* has no input parameters.

2) For the null-call method *ST_StartM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateStartM* of SELF.

3) The method *ST_StartM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *astartm*.

4) For the type-preserving method *ST_StartM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0 (zero).

      ii) If *SELF.ST_PrivateEndM* is NOT NULL, set *M* = 1 (one).

      iii) If *astartm* IS NULL, set *M* to 0 (zero).

iv) Return an *ST_NURBSCurve* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_StartM* attribute set to *astartm*.

7.7.7 ST_EndM Methods

Purpose

Observe and mutate the attribute ST_PrivateEndM of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndM
      END

CREATE METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_NURBSCurve
   FOR ST_NURBSCurve
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_NURBSCurve);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateStartM is NOT NULL THEN
         SET measured = 1;
      -- If anendm is NULL, IS_Measured must be 0 (zero)
      IF anendm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateEndM(anendm);
   END
```

Description

1) The method *ST_EndM()* has no input parameters.

2) For the null-call method *ST_EndM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateEndM* of SELF.

3) The method *ST_EndM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *anendm*.

4) For the type-preserving method *ST_EndM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0 (zero).

      ii) If *SELF.ST_PrivateStartM* is NOT NULL, set *M* = 1 (one).

      iii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *SP = SP.ST_M(SELF.ST_StartM())*.

iv) If *anendm* IS NULL, set *M* to 0 (zero).

v) Return an *ST_NURBSCurve* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_EndM* attribute set to *anendm*.

7.7.8 ST_StartPoint Method

Purpose

Return the start point of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.

2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *SP* be the *ST_Point* start point value calculated from the *ST_PrivateDegree, ST_PrivateControlPoints* and *ST_PrivateKnots* attribute values of SELF.

      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *SP = SP.ST_M(SELF.ST_StartM())*.

      iii) Return *SP*.

7.7.9 ST_EndPoint Method

Purpose

Return the end point of an ST_NURBSCurve value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_NURBSCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.

2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *EP* be the *ST_Point* end point value calculated from the *ST_PrivateDegree, ST_PrivateControlPoints* and *ST_PrivateKnots* attribute values of SELF.

      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *EP* = *EP.ST_M(SELF.ST_EndM())*.

      iii) Return *EP*.

7.7.10 ST_NURBSFromTxt Functions

Purpose

Return an ST_NURBSCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_NURBSCurve value.

Definition

```
CREATE FUNCTION ST_NURBSFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_NURBSFromTxt(awkt, 0)

CREATE FUNCTION ST_NURBSFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_NURBSFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_NURBSFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_NURBSFromTxt(awkt, 0)*.
3) The function *ST_NURBSFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_NURBSFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_NURBSCurve* value.

      If *awkt* is not producible in the BNF for <nurbs text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_NURBSCurve)*.

7.7.11 ST_NURBSFromWKB Functions

Purpose

Return an ST_NURBSCurve value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_NURBSCurve value.

Definition

```
CREATE FUNCTION ST_NURBSFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_NURBSFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_NURBSFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_NURBSFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_NURBSFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_NURBSFromWKB(awkb, 0)*.
3) The function *ST_NURBSFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_NURBSFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_NURBSCurve* value.

      If *awkb* is not producible in the BNF for <nurbs binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_NURBSCurve)*.

7.7.12 ST_NURBSFromGML Functions

Purpose

Return an ST_NURBSCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML BSpline representation of an ST_NURBSCurve value.

Definition

```
CREATE FUNCTION ST_NURBSFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_NURBSFromGML(agml, 0)

CREATE FUNCTION ST_NURBSFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_NURBSCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_NURBSFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_NURBSFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_NURBSFromGML(agml, 0)*.
3) The function *ST_NURBSFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_NURBSFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a BSpline XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_NURBSCurve)*.

7.8 ST_Clothoid Type and Routines

7.8.1 ST_Clothoid Type

Purpose

The ST_Clothoid type is a subtype of the ST_Curve type and represents a single curve segment having clothoid interpolation.

Definition

```
CREATE TYPE ST_Clothoid
   UNDER ST_Curve
   AS (
      ST_PrivateReferenceLocation ST_AffinePlacement DEFAULT NULL,
      ST_PrivateScaleFactor DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateStartDistance DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateEndDistance DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateStartM DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateEndM DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Clothoid
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_Clothoid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Clothoid
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_Clothoid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Clothoid
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_Clothoid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Clothoid
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_Clothoid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_RefLocation()
   RETURNS ST_AffinePlacement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_RefLocation
   (areflocation ST_AffinePlacement)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ScaleFactor()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ScaleFactor
   (ascalefactor DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartDistance()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartDistance
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartDistance
   (astartdistance DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_StartDistance
   (astartdistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EndDistance()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndDistance
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndDistance
   (anenddistance DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EndDistance
   (anenddistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
5) The attribute *ST_PrivateReferenceLocation* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferenceLocation*.
6) The attribute *ST_PrivateScaleFactor* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateScaleFactor*.
7) The attribute *ST_PrivateStartDistance* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartDistance*.
8) The attribute *ST_PrivateEndDistance* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndDistance*.
9) The attribute *ST_PrivateStartM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartM*.
10) The attribute *ST_PrivateEndM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndM*.

Description

1) The *ST_Clothoid* type provides for public use:
   a) a method *ST_Clothoid(CHARACTER LARGE OBJECT)*,
   b) a method *ST_Clothoid(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_Clothoid(BINARY LARGE OBJECT)*,
   d) a method *ST_Clothoid(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)*,

f) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,

g) a method ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION),

h) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,

i) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)*,

j) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*,

k) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)*,

l) a method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*,

m) a method *ST_RefLocation()*,

n) a method *ST_RefLocation(ST_AffinePlacement)*,

o) a method *ST_ScaleFactor()*,

p) a method *ST_ScaleFactor(DOUBLE PRECISION)*,

q) a method *ST_StartDistance()*,

r) a method *ST_StartDistance(CHARACTER VARYING)*,

s) a method *ST_StartDistance(DOUBLE PRECISION)*,

t) a method *ST_StartDistance(DOUBLE PRECISION, CHARACTER VARYING)*,

u) a method *ST_EndDistance()*,

v) a method *ST_EndDistance(CHARACTER VARYING)*,

w) a method *ST_EndDistance(DOUBLE PRECISION)*,

x) a method *ST_EndDistance(DOUBLE PRECISION, CHARACTER VARYING)*,

y) a method *ST_StartM()*,

z) a method *ST_StartM(DOUBLE PRECISION)*,

aa) a method *ST_EndM()*,

ab) a method *ST_EndM(DOUBLE PRECISION)*,

ac) an overriding method *ST_StartPoint()*,

ad) an overriding method *ST_EndPoint()*,

ae) a function *ST_ClothoidFromTxt(CHARACTER LARGE OBJECT)*,

af) a function *ST_ClothoidFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,

ag) a function *ST_ClothoidFromWKB(BINARY LARGE OBJECT)*,

ah) a function *ST_ClothoidFromWKB(BINARY LARGE OBJECT, INTEGER)*,

ai) a function *ST_ClothoidFromGML(CHARACTER LARGE OBJECT)*,

aj) a function *ST_ClothoidFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateReferenceLocation* attribute contains the *ST_AffinePlacement* reference location value.

3) The *ST_PrivateScaleFactor* attribute contains the DOUBLE PRECISION scale factor value.

4) The *ST_PrivateStartDistance* attribute contains the DOUBLE PRECISION start distance value.

5) The *ST_PrivateEndDistance* attribute contains the DOUBLE PRECISION end distance value.
6) The *ST_PrivateStartM* attribute contains the DOUBLE PRECISION start measure value.
7) The *ST_PrivateEndM* attribute contains the DOUBLE PRECISION end measure value.
8) An *ST_Clothoid* is not well formed if *ST_PrivateStartM* is NULL and *ST_PrivateEndM* is NOT NULL or if *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NULL.
9) If *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NOT NULL, then *SELF.ST_IsMeasured* equals 1 (one). Otherwise, *SELF.ST_IsMeasured* equals 0 (zero).
10) The *ST_Clothoid ST_PrivateIs3D* attribute value shall be equal to the *ST_PrivateIs3D* attribute value of the *ST_PrivateReferenceLocation ST_AffinePlacement ST_Point ST_PrivateLocation* attribute value.
11) Let *AV* be the *ST_PrivateReferenceDirections* attribute *ST_Vector* ARRAY value of the *ST_Clothoid ST_PrivateReferenceLocation* attribute *ST_AffinePlacement* value. Then the *ST_Clothoid ST_PrivateIs3D* value shall be equal to one (1) if and only if the value returned by the *ST_GetCoordDim(AV)* function is equal to 3.
12) The cardinality of the *ST_PrivateReferenceLocation ST_AffinePlacement ST_Vector* ARRAY *ST_PrivateReferenceDirections* attribute value shall be equal to 2.
13) The type *ST_Clothoid* is a subtype of *ST_Curve* with clothoid interpolation.
14) The type *ST_Clothoid* defines a single clothoid curve *segment*.
15) If *SELF.ST_IsMeasured* equals 1 (one), then the *ST_PrivateStartM* and ST_*PrivateEndM* m coordinate values shall be included in the start and endpoint coordinates for *ST_StartPoint* and *ST_EndPoint*, respectively.

7.8.2 ST_Clothoid Methods

Purpose

Return an ST_Clothoid value constructed from either the well-known text representation; the well-known binary representation; the GML representation; or the specified reference location ST_AffinePlacement value and DOUBLE PRECISION scale factor, start distance, and end distance values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Clothoid
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN ST_ClothoidFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(areferencelocation, ascalefactor,
     astartdistance, anenddistance, NULL, NULL, 0)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(areferencelocation, ascalefactor,
      astartdistance, anenddistance, NULL, NULL, ansrid)
```

```
CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(areferencelocation, ascalefactor,
     astartdistance, anenddistance, astartm, anendm, 0)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(areferencelocation, ascalefactor,
     astartdistance, anenddistance, NULL, NULL, aunit, 0)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(areferencelocation, ascalefactor,
      astartdistance, anenddistance, NULL, NULL, aunit, ansrid)

CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   RETURN NEW ST_Clothoid(areferencelocation, ascalefactor,
      astartdistance, anenddistance, astartm, anendm, aunit, 0)
```

```
CREATE CONSTRUCTOR METHOD ST_Clothoid
   (areferencelocation ST_AffinePlacement,
    ascalefactor DOUBLE PRECISION,
    astartdistance DOUBLE PRECISION,
    anenddistance DOUBLE PRECISION,
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Geometry* value.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Clothoid(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_Clothoid(awktorgml, 0)*.
3) The method *ST_Clothoid(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_Clothoid(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If *awktorgml* contains a Clothoid XML element in the GML representation, then return the result of the value expression: *ST_ClothoidFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_ClothoidFromTxt(awktorgml, ansrid)*.
5) The method *ST_Clothoid(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_Clothoid(awkb, 0)*.
7) The method *ST_Clothoid(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_ClothoidFromWKB(awkb, ansrid)*.

9) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

   a) an *ST_AffinePlacement* value *areferencelocation,*

   b) a DOUBLE PRECISION value *ascalefactor,*

   c) a DOUBLE PRECISION value *astartdistance,*

   d) a DOUBLE PRECISION value *anenddistance.*

10) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Clothoid(areferencelocation, ascalefactor, astartdistance, anenddistance, NULL, NULL, 0)*.

11) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) an *ST_AffinePlacement* value *areferencelocation,*

   b) a DOUBLE PRECISION value *ascalefactor,*

   c) a DOUBLE PRECISION value *astartdistance,*

   d) a DOUBLE PRECISION value *anenddistance,*

   e) an INTEGER value *ansrid.*

12) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* returns the result of the value expression: *NEW ST_Clothoid(areferencelocation, ascalefactor, astartdistance, anenddistance, NULL, NULL, ansrid)*.

13) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

   a) an *ST_AffinePlacement* value *areferencelocation*,

   b) a DOUBLE PRECISION value *ascalefactor,*

   c) a DOUBLE PRECISION value *astartdistance,*

   d) a DOUBLE PRECISION value *anenddistance,*

   e) a DOUBLE PRECISION value *astartm,*

   f) a DOUBLE PRECISION value *anendm*.

14) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Clothoid(areferencelocation, ascalefactor, astartdistance, anenddistance, astartm, anendm, 0)*.

15) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) an *ST_AffinePlacement* value *areferencelocation*,

   b) a DOUBLE PRECISION value *ascalefactor,*

   c) a DOUBLE PRECISION value *astartdistance,*

   d) a DOUBLE PRECISION value *anenddistance,*

   e) a DOUBLE PRECISION value *astartm,*

   f) a DOUBLE PRECISION value *anendm*,

g) an INTEGER value *ansrid.*

16) Case:

a) If the spatial reference system *ansrid* defines a <linear unit>, then the values *astartdistance* and *anenddistance* are in the linear unit of measure identified by <linear unit>.

b) Otherwise, the values *astartdistance* and *anenddistance* are in an implementation-defined unit of measure.

17) For the type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

Case:

a) If *areferencelocation* is the null value or if *ascalefactor* is the null value or if *astartdistance* is the null value or if *anenddistance* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_Clothoid* value with:

i) The *ST_PrivateDimension* attribute set to 1 (one).

ii) Case:

1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

iii) The *ST_PrivateCoordinateDimension* attribute set to *areferencelocation.ST_Location.ST_CoordDim* + *ST_PrivateIsMeasured*.

iv) The *ST_PrivateIs3D* attribute set to *areferencelocation.ST_Location.ST_Is3D*.

v) Case:

1) If CARDINALITY(*areferencelocation.ST_RefDirections()*) not equal to 2, then an exception is raised: *SQL/MM Spatial exception – incorrect number of vectors*.

2) Otherwise, the *ST_PrivateReferenceLocation* attribute set to *areferencelocation*.

vi) The *ST_PrivateScaleFacto*r attribute set to *ascalefactor.*

vii) The *ST_PrivateStartDistance* attribute set to *astartdistance*.

viii) The *ST_PrivateEndDistance* attribute set to *anenddistance*.

ix) The *ST_PrivateStartM* attribute set to *astartm*.

x) The *ST_PrivateEndM* attribute set to *anendm*.

xi) The spatial reference system identifier set to *ansrid*.

18) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *ascalefactor,*

c) a DOUBLE PRECISION value *astartdistance,*

d) a DOUBLE PRECISION value *anenddistance,*

e) a CHARACTER VARYING value *aunit.*

19) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_Clothoid(areferencelocation, ascalefactor, astartdistance, anenddistance, NULL, NULL, aunit, 0)*.

20) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* takes the following input parameters:

    a) an *ST_AffinePlacement* value *areferencelocation,*

    b) a DOUBLE PRECISION value *ascalefactor,*

    c) a DOUBLE PRECISION value *astartdistance,*

    d) a DOUBLE PRECISION value *anenddistance,*

    e) a CHARACTER VARYING value *aunit,*

    f) an INTEGER value *ansrid.*

21) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* returns the result of the value expression: *NEW ST_Clothoid(areferencelocation, ascalefactor, astartdistance, anenddistance, NULL, NULL, aunit, ansrid)*.

22) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

    a) an *ST_AffinePlacement* value *areferencelocation*,

    b) a DOUBLE PRECISION value *ascalefactor,*

    c) a DOUBLE PRECISION value *astartdistance,*

    d) a DOUBLE PRECISION value *anenddistance,*

    e) a DOUBLE PRECISION value *astartm,*

    f) a DOUBLE PRECISION value *anendm*,

    g) a CHARACTER VARYING value *aunit.*

23) The null-call type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_Clothoid(areferencelocation, ascalefactor, astartdistance, anenddistance, astartm, astartm, aunit, 0)*.

24) The method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* takes the following input parameters:

    a) an *ST_AffinePlacement* value *areferencelocation*,

    b) a DOUBLE PRECISION value *ascalefactor,*

    c) a DOUBLE PRECISION value *astartdistance,*

    d) a DOUBLE PRECISION value *anenddistance,*

    e) a DOUBLE PRECISION value *astartm,*

    f) a DOUBLE PRECISION value *anendm*,

    g) a CHARACTER VARYING value *aunit,*

    h) an INTEGER value *ansrid.*

25) For the values astartdistance and anenddistance:

    a) The value for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

26) For the type-preserving SQL-invoked constructor method *ST_Clothoid(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*:

Case:

a) If *areferencelocation* is the null value or if *ascalefactor* is the null value or if *astartdistance* is the null value or if *anenddistance* is the null value or if *aunit* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_Clothoid* value with:

i) The *ST_PrivateDimension* attribute set to 1 (one).

ii) Case:

1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

iii) The *ST_PrivateCoordinateDimension* attribute set to *areferencelocation.ST_Location.ST_CoordDim* + *ST_PrivateIsMeasured*.

iv) The *ST_PrivateIs3D* attribute set to *areferencelocation.ST_Location.ST_Is3D*.

v) Case:

1) If CARDINALITY(areferencelocation.ST_RefDirections()) not equal to 2, then an exception is raised: SQL/MM Spatial exception – incorrect number of vectors.

2) Otherwise, the *ST_PrivateReferenceLocation* attribute set to *areferencelocation*.

vi) The *ST_PrivateScaleFacto*r attribute set to *ascalefactor.*

vii) The *ST_PrivateStartDistance* attribute set to *astartdistance*.

viii) The *ST_PrivateEndDistance* attribute set to *anenddistance*.

ix) The *ST_PrivateStartM* attribute set to *astartm*.

x) The *ST_PrivateEndM* attribute set to *anendm*.

xi) The spatial reference system identifier set to *ansrid*.

7.8.3 ST_RefLocation Methods

Purpose

Observe and mutate the attribute ST_PrivateReferenceLocation of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_RefLocation()
   RETURNS ST_AffinePlacement
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateReferenceLocation
      END

CREATE METHOD ST_RefLocation
   (areflocation ST_AffinePlacement)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      -- If areflocation is the null value, then raise an exception
      IF areflocation IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      -- Check if curve and affine placement location point are both 2D
      -- or both 3D
      IF SELF.ST_Is3D() <> areflocation.ST_Location().ST_Is3D() THEN
         SIGNAL SQLSTATE '2FF96'
            SET MESSAGE_TEXT = 'mixed Is3D';
      END IF;
      -- Check if affine placement reference directions has two vectors
      IF CARDINALITY(areflocation.ST_RefDirections()) <> 2 THEN
         SIGNAL SQLSTATE '2FF86'
            SET MESSAGE_TEXT = 'incorrect number of vectors';
      END IF;
      RETURN
         SELF.ST_PrivateReferenceLocation(areflocation);
   END
```

Description

1) The method *ST_RefLocation()* has no input parameters.
2) For the null-call method *ST_RefLocation()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateReferenceLocation* of SELF.
3) The method *ST_RefLocation(ST_AffinePlacement)* takes the following input parameters:
   a) an *ST_AffinePlacement* value *areflocation*.
4) For the type-preserving method *ST_RefLocation(ST_AffinePlacement)*:

   Case:

a) If *areflocation* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) If SELF and the *ST_PrivateLocation* attribute *ST_Point* value of *areflocation* are not either both 2D or both 3D, then an exception condition is raised: *SQL/MM Spatial exception – mixed Is3D*.

d) If the number of *ST_Vectors* in the *ST_PrivateReferenceDirections* attribute of *areflocation* is not equal to 2, then an exception condition is raised: *SQL/MM Spatial exception – incorrect number of vectors*.

e) Otherwise, return an *ST_Clothoid* value with the attribute *ST_PrivateReferenceLocation* set to *areflocation*.

7.8.4 ST_ScaleFactor Methods

Purpose

Observe and mutate the attribute ST_PrivateScaleFactor of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_ScaleFactor()
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateScaleFactor
      END

CREATE METHOD ST_ScaleFactor
   (ascalefactor DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      -- If ascalefactor is the null value, then raise an exception
      IF ascalefactor IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      RETURN
         SELF.ST_PrivateScaleFactor(ascalefactor);
   END
```

Description

1) The method *ST_ScaleFactor()* has no input parameters.
2) For the null-call method *ST_ScaleFactor()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateScaleFactor* of SELF.
3) The method *ST_ScaleFactor(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *ascalefactor*.
4) For the type-preserving method *ST_ScaleFactor(DOUBLE PRECISION)*:

   Case:

   a) If *ascalefactor* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_Clothoid* value with the attribute *ST_PrivateScaleFactor* set to *ascalefactor*.

7.8.5 ST_StartDistance Methods

Purpose

Observe and mutate the attribute ST_PrivateStartDistance of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_StartDistance()
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartDistance
      END

CREATE METHOD ST_StartDistance
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartDistance
      END

CREATE METHOD ST_StartDistance
   (astartdistance DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      -- If astartdistance is the null value, then raise an exception
      IF astartdistance IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      RETURN
         SELF.ST_PrivateStartDistance(astartdistance);
   END
```

```
CREATE METHOD ST_StartDistance
   (astartdistance DOUBLE PRECISION,
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      -- If astartdistance is the null value, then raise an exception
      IF astartdistance IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      RETURN
         SELF.ST_PrivateStartDistance(astartdistance);
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_StartDistance()* has no input parameters.
2) For the null-call method *ST_StartDistance()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateStartDistance* of SELF.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_StartDistance()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_StartDistance()* is in an implementation-defined unit of measure.
3) The method *ST_StartDistance(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_StartDistance(CHARACTER VARYING)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateStartDistance* of SELF.
5) For the method *ST_StartDistance(CHARACTER VARYING)*:
   a) The value for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
6) The method *ST_StartDistance(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *astartdistance*.

7) For the type-preserving method *ST_StartDistance(DOUBLE PRECISION)*:

   Case:

   a) If *astartdistance* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_Clothoid* value with the attribute *ST_PrivateStartDistance* set to *astartdistance*.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_StartDistance()* is in the linear unit of measure identified by <linear unit>.

      ii) Otherwise, the value returned by *ST_StartDistance()* is in an implementation-defined unit of measure.

8) The method *ST_StartDistance(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

   a) a DOUBLE PRECISION value *astartdistance*,

   b) a CHARACTER VARYING value *aunit*.

9) For the type-preserving method *ST_StartDistance(DOUBLE PRECISION, CHARACTER VARYING)*:

   Case:

   a) If *astartdistance* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_Clothoid* value with the attribute *ST_PrivateStartDistance* set to *astartdistance*.

10) For the method *ST_StartDistance(DOUBLE PRECISION, CHARACTER VARYING)*:

   a) The value for *aunit* shall be a supported <unit name>.

   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

7.8.6 ST_EndDistance Methods

Purpose

Observe and mutate the attribute ST_PrivateEndDistance of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_EndDistance()
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndDistance
      END

CREATE METHOD ST_EndDistance
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndDistance
      END

CREATE METHOD ST_EndDistance
   (anenddistance DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      -- If anenddistance is the null value, then raise an exception
      IF anenddistance IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      RETURN
         SELF.ST_PrivateEndDistance(anenddistance);
   END
```

```
CREATE METHOD ST_EndDistance
   (anenddistance DOUBLE PRECISION,
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      -- If anenddistance is the null value, then raise an exception
      IF anenddistance IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      RETURN
         SELF.ST_PrivateEndDistance(anenddistance);
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_EndDistance()* has no input parameters.
2) For the null-call method *ST_EndDistance()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateEndDistance* of SELF.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_EndDistance()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_EndDistance()* is in an implementation-defined unit of measure.
3) The method *ST_EndDistance(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_EndDistance(CHARACTER VARYING)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateEndDistance* of SELF.
5) For the method *ST_EndDistance(CHARACTER VARYING)*:
   a) The value for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
6) The method *ST_EndDistance(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *anenddistance*.

7) For the type-preserving method *ST_EndDistance(DOUBLE PRECISION)*:

   Case:

   a) If *anenddistance* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_Clothoid* value with the attribute *ST_PrivateEndDistance* set to *anenddistance*.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_EndDistance()* is in the linear unit of measure identified by <linear unit>.

      ii) Otherwise, the value returned by *ST_EndDistance()* is in an implementation-defined unit of measure.

8) The method *ST_EndDistance(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

   a) a DOUBLE PRECISION value *anenddistance*,

   b) a CHARACTER VARYING value *aunit*.

9) For the type-preserving method *ST_EndDistance(DOUBLE PRECISION, CHARACTER VARYING)*:

   Case:

   a) If *anenddistance* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_Clothoid* value with the attribute *ST_PrivateEndDistance* set to *anenddistance*.

10) For the method *ST_EndDistance(DOUBLE PRECISION, CHARACTER VARYING)*:

   a) The value for *aunit* shall be a supported <unit name>.

   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

7.8.7 ST_StartM Methods

Purpose

Observe and mutate the attribute ST_PrivateStartM of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartM
      END

CREATE METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateEndM is NOT NULL THEN
         SET measured = 1;
      -- If astartm is NULL, IS_Measured must be 0 (zero)
      IF astartm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateStartM(astartm);
   END
```

Description

1) The method *ST_StartM()* has no input parameters.

2) For the null-call method *ST_StartM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateStartM* of SELF.

3) The method *ST_StartM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *astartm*.

4) For the type-preserving method *ST_StartM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0 (zero).

      ii) If *SELF.ST_PrivateEndM* is NOT NULL, set *M* = 1 (one).

      iii) If *astartm* IS NULL, set *M* to 0 (zero).

iv) Return an *ST_Clothoid* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_StartM* attribute set to *astartm*.

7.8.8 ST_EndM Methods

Purpose

Observe and mutate the attribute ST_PrivateEndM of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndM
      END

CREATE METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   FOR ST_Clothoid
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Clothoid);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateStartM is NOT NULL THEN
         SET measured = 1;
      -- If anendm is NULL, IS_Measured must be 0 (zero)
      IF anendm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateEndM(anendm);
   END
```

Description

1) The method *ST_EndM()* has no input parameters.

2) For the null-call method *ST_EndM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateEndM* of SELF.

3) The method *ST_EndM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *anendm*.

4) For the type-preserving method *ST_EndM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0 (zero).

      ii) If *SELF.ST_PrivateStartM* is NOT NULL, set *M* = 1 (one).

      iii) If *anendm* IS NULL, set *M* to 0 (zero).

iv) Return an *ST_Clothoid* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_EndM* attribute set to *anendm*.

7.8.9 ST_StartPoint Method

Purpose

Return the start point of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.

2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *SP* be the *ST_Point* start point value calculated from the *ST_PrivateReferenceLocation, ST_PrivateScaleFactor , ST_PrivateStartDistance,* and *ST_PrivateEndDistance* attribute values of SELF.

      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *SP = SP.ST_M(SELF.ST_StartM())*.

      iii) Return *SP*.

7.8.10 ST_EndPoint Method

Purpose

Return the end point of an ST_Clothoid value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_Clothoid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.
2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Let *EP* be the *ST_Point* end point value calculated from the *ST_PrivateReferenceLocation, ST_PrivateScaleFactor , ST_PrivateStartDistance,* and *ST_PrivateEndDistance* attribute values of SELF.
      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *EP = EP.ST_M(SELF.ST_EndM())*.
      iii) Return *EP*.

7.8.11 ST_ClothoidFromTxt Functions

Purpose

Return an ST_Clothoid value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Clothoid value.

Definition

```
CREATE FUNCTION ST_ClothoidFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_ClothoidFromTxt(awkt, 0)
```

```
CREATE FUNCTION ST_ClothoidFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_ClothoidFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_ClothoidFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_ClothoidFromTxt(awkt, 0)*.
3) The function *ST_ClothoidFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_ClothoidFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Clothoid* value.

      If *awkt* is not producible in the BNF for <clothoid text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_Clothoid)*.

7.8.12 ST_ClothoidFromWKB Functions

Purpose

Return an ST_Clothoid value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Clothoid value.

Definition

```
CREATE FUNCTION ST_ClothoidFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_ClothoidFromWKB(awkb, 0)

CREATE FUNCTION ST_ClothoidFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_ClothoidFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_ClothoidFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_ClothoidFromWKB(awkb, 0)*.
3) The function *ST_ClothoidFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_ClothoidFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_Clothoid* value.

      If *awkb* is not producible in the BNF for <clothoid binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_Clothoid)*.

7.8.13 ST_ClothoidFromGML Functions

Purpose

Return an ST_Clothoid value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Clothoid representation of an ST_Clothoid value.

Definition

```
CREATE FUNCTION ST_ClothoidFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_ClothoidFromGML(agml, 0)

CREATE FUNCTION ST_ClothoidFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_Clothoid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_ClothoidFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_ClothoidFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_ClothoidFromGML(agml, 0)*.
3) The function *ST_ClothoidFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_ClothoidFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain an Clothoid XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_Clothoid)*.

7.9 ST_SpiralCurve Type and Routines

7.9.1 ST_SpiralCurve Type

Purpose

The ST_SpiralCurve type is a subtype of the ST_Curve type and represents a single curve segment having spiral interpolation.

Definition

```
CREATE TYPE ST_SpiralCurve
   UNDER ST_Curve
   AS (
      ST_PrivateReferenceLocation ST_AffinePlacement DEFAULT NULL,
      ST_PrivateLength DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateStartCurvature DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateEndCurvature DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateSpiralType CHARACTER VARYING(64) DEFAULT NULL,
      ST_PrivateStartM DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateEndM DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_SpiralCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_SpiralCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_SpiralCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_SpiralCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_SpiralCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_SpiralCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_SpiralCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64))
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_RefLocation()
   RETURNS ST_AffinePlacement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_RefLocation
   (areflocation ST_AffinePlacement)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Length
   (alength DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Length
   (alength DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartCurvature()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_StartCurvature
   (astartcurvature DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EndCurvature()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndCurvature
   (anendcurvature DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_SpiralType()
   RETURNS CHARACTER VARYING(64)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SpiralType
   (asspiraltype CHARACTER VARYING(64))
   RETURNS ST_SpiralCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_Clothoid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

OVERRIDING METHOD ST_Length()
   RETURNS DOUBLE PRECISION,

OVERRIDING METHOD ST_Length
   (CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
5) The attribute *ST_PrivateReferenceLocation* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferenceLocation*.
6) The attribute *ST_PrivateLength* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLength*.
7) The attribute *ST_PrivateStartCurvature* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartCurvature*.
8) The attribute *ST_PrivateEndCurvature* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndCurvature*.
9) The attribute *ST_PrivateSpiralType* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateSpiralType*.
10) The attribute *ST_PrivateStartM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartM*.
11) The attribute *ST_PrivateEndM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEndM*.

Description

1) The *ST_SpiralCurve* type provides for public use:

a) a method *ST_SpiralCurve(CHARACTER LARGE OBJECT)*,

b) a method *ST_SpiralCurve(CHARACTER LARGE OBJECT, INTEGER)*,

c) a method *ST_SpiralCurve(BINARY LARGE OBJECT)*,

d) a method *ST_SpiralCurve(BINARY LARGE OBJECT, INTEGER)*,

e) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)*,

f) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*,

g) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION)*,

h) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,

i) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, CHARACTER VARYING)*,

j) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, CHARACTER VARYING, INTEGER)*,

k) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)*,

l) a method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*,

m) a method *ST_RefLocation()*,

n) a method *ST_RefLocation(ST_AffinePlacement)*,

o) a method *ST_Length(DOUBLE PRECISION)*,

p) a method *ST_Length(DOUBLE PRECISION, CHARACTER VARYING)*,

q) a method *ST_StartCurvature()*,

r) a method *ST_StartCurvature(DOUBLE PRECISION)*,

s) a method *ST_EndCurvature()*,

t) a method *ST_EndCurvature(DOUBLE PRECISION)*,

u) a method *ST_SpiralType()*,

v) a method *ST_SpiralType(CHARACTER VARYING)*,

w) a method *ST_StartM()*,

x) a method *ST_StartM(DOUBLE PRECISION)*,

y) a method *ST_EndM()*,

z) a method *ST_EndM(DOUBLE PRECISION)*,

aa) an overriding method *ST_Length()*,

ab) an overriding method *ST_Length(CHARACTER VARYING)*,

ac) an overriding method *ST_StartPoint()*,

ad) an overriding method *ST_EndPoint()*,

ae) a function *ST_SpiralFromTxt(CHARACTER LARGE OBJECT)*,

af) a function *ST_SpiralFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,

ag) a function *ST_SpiralFromWKB(BINARY LARGE OBJECT)*,

ah) a function *ST_SpiralFromWKB(BINARY LARGE OBJECT, INTEGER)*,

ai) a function *ST_SpiralFromGML(CHARACTER LARGE OBJECT)*,

aj) a function *ST_SpiralFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateReferenceLocation* attribute contains the *ST_AffinePlacement* reference location value.

3) The *ST_PrivateLength* attribute contains the DOUBLE PRECISION length value.

4) The *ST_PrivateStartCurvature* attribute contains the DOUBLE PRECISION start curvature value.

5) The *ST_PrivateEndCurvature* attribute contains the DOUBLE PRECISION end curvature value.

6) The *ST_PrivateSpiralType* attribute contains the CHARACTER VARYING spiral type value.

7) The *ST_PrivateStartM* attribute contains the DOUBLE PRECISION start measure value.

8) The *ST_PrivateEndM* attribute contains the DOUBLE PRECISION end measure value.

9) An *ST_SpiralCurve* is not well formed if *ST_PrivateStartM* is NULL and *ST_PrivateEndM* is NOT NULL or if *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NULL.

10) If *ST_PrivateStartM* is NOT NULL and *ST_PrivateEndM* is NOT NULL, then *SELF.ST_IsMeasured* equals 1 (one). Otherwise, *SELF.ST_IsMeasured* equals 0 (zero).

11) The *ST_SpiralCurve ST_PrivateIs3D* attribute value shall be equal to the *ST_PrivateIs3D* attribute value of the *ST_PrivateReferenceLocation ST_AffinePlacement ST_Point ST_PrivateLocation* attribute value.

12) Let *AV* be the *ST_PrivateReferenceDirections* attribute*ST_Vector* ARRAY value of the *ST_SpiralCurve ST_PrivateReferenceLocation* attribute *ST_AffinePlacement* value. Then the *ST_SpiralCurve ST_PrivateIs3D* value shall be equal to one (1) if and only if the value returned by the *ST_GetCoordDim(AV)* function is equal to 3.

13) The cardinality of the *ST_PrivateReferenceLocation ST_AffinePlacement ST_Vector* ARRAY *ST_PrivateReferenceDirections* attribute value shall be equal to 2.

14) The type *ST_SpiralCurve* is a subtype of *ST_Curve* with spiral interpolation.

15) The type *ST_SpiralCurve* defines a single Spiral curve *segment*.

16) An *ST_SpiralCurve* value returned by the constructor function corresponds to the empty set.

17) Curvature is one over the radius of curvature.

18) If *SELF.ST_IsMeasured* equals 1 (one), then the *ST_PrivateStartM* and ST*_PrivateEndM* m coordinate values shall be included in the start and endpoint coordinates for *ST_StartPoint* and *ST_EndPoint*, respectively.

19) The curvature function for the specific spiral type shall be implementation-defined.

7.9.2 ST_SpiralCurve Methods

Purpose

Return an ST_SpiralCurve value constructed from either the well-known text representation; the well-known binary representation; the GML representation; or the specified reference location ST_AffinePlacement value, DOUBLE PRECISION length, start curvature, and end curvature values, and the CHARACTER VARYING spiral type value.

Definition

```
CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN ST_SpiralFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(areferencelocation, alength,
      astartcurvature, anendcurvature, aspiraltype, NULL, NULL, 0)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(areferencelocation, alength,
      astartcurvature, anendcurvature, aspiraltype, NULL, NULL, ansrid)
```

```
CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(areferencelocation, alength,
     astartcurvature, anendcurvature, aspiraltype, astartm, anendm, 0)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(areferencelocation, alength,
      astartcurvature, anendcurvature, aspiraltype, NULL, NULL, aunit, 0)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(areferencelocation, alength,
      astartcurvature, anendcurvature, aspiraltype, NULL, NULL, aunit,
      ansrid)
```

```
CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   RETURN NEW ST_SpiralCurve(areferencelocation, alength,
      astartcurvature, anendcurvature, aspiraltype, astartm, anendm,
      aunit, 0)

CREATE CONSTRUCTOR METHOD ST_SpiralCurve
   (areferencelocation ST_AffinePlacement,
    alength DOUBLE PRECISION,
    astartcurvature DOUBLE PRECISION,
    anendcurvature DOUBLE PRECISION,
    aspiraltype CHARACTER VARYING(64),
    astartm DOUBLE PRECISION,
    anendm DOUBLE PRECISION,
    aunit CHARACTER VARYING(ST_MaxUnitNameLength),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Geometry* value.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_SpiralCurve(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_SpiralCurve(awktorgml, 0)*.
3) The method *ST_SpiralCurve(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

a) If *awktorgml* contains a SpiralCurve XML element in the GML representation, then return the result of the value expression: *ST_SpiralFromGML(awktorgml, ansrid)*.

b) Otherwise, return the result of the value expression: *ST_SpiralFromTxt(awktorgml, ansrid)*.

5) The method *ST_SpiralCurve(BINARY LARGE OBJECT)* takes the following input parameter:

a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_SpiralCurve(awkb, 0)*.

7) The method *ST_SpiralCurve(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a BINARY LARGE OBJECT value *awkb*,

b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_SpiralFromWKB(awkb, ansrid)*.

9) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*.

10) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_SpiralCurve(areferencelocation, alength, astartcurvature, anendcurvature, aspiraltype, NULL, NULL, 0)*.

11) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*,

f) an INTEGER value *ansrid.*

12) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* returns the result of the value expression: *NEW ST_SpiralCurve(areferencelocation, alength, astartcurvature, anendcurvature, aspiraltype, NULL, NULL, ansrid)*.

13) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype,*

f) a DOUBLE PRECISION value *astartm,*

g) a DOUBLE PRECISION value *anendm*.

14) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_SpiralCurve(areferencelocation, alength, astartcurvature, anendcurvature, aspiraltype, astartm, anendm, 0)*.

15) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*,

f) a DOUBLE PRECISION value *astartm,*

g) a DOUBLE PRECISION value *anendm*,

h) an INTEGER value *ansrid.*

16) Case:

a) If the spatial reference system *ansrid* defines a <linear unit>, then the value *alength* is in the linear unit of measure identified by <linear unit>.

b) Otherwise, the value *alength* is in an implementation-defined unit of measure.

17) For the type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

Case:

a) If *areferencelocation* is the null value or if *alength* is the null value or if *astartcurvature* is the null value or if *anendcurvature* is the null value or if *aspiraltype* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_SpiralCurve* value with:

i) The *ST_PrivateDimension* attribute set to 1 (one).

ii) Case:

1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

iii) The *ST_PrivateCoordinateDimension* attribute set to *areferencelocation.ST_Location.ST_CoordDim* + *ST_PrivateIsMeasured*.

iv) The *ST_PrivateIs3D* attribute set to *areferencelocation.ST_Location.ST_Is3D.*

v) Case:

1) If CARDINALITY(*areferencelocation.ST_RefDirections()*) not equal to 2, then an exception is raised: *SQL/MM Spatial exception – incorrect number of vectors*.

2) Otherwise, the *ST_PrivateReferenceLocation* attribute set to *areferencelocation*.

vi) The *ST_PrivateLength* attribute set to *alength*.

vii) The *ST_PrivateStartCurvature* attribute set to *astartcurvature*.

viii) The *ST_PrivateEndCurvature* attribute set to *anendcurvature*.

ix) The *ST_PrivateSpiralType* attribute set to *aspiraltype*.

x) The *ST_PrivateStartM* attribute set to *astartm*.

xi) The *ST_PrivateEndM* attribute set to *anendm*.

xii) The spatial reference system identifier set to *ansrid*.

18) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*,

f) a CHARACTER VARYING value *aunit.*

19) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_SpiralCurve(areferencelocation, alength, astartcurvature, anendcurvature, aspiraltype, NULL, NULL, aunit, 0)*.

20) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, CHARACTER VARYING, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation,*

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*,

f) a CHARACTER VARYING value *aunit,*

g) an INTEGER value *ansrid.*

21) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* returns the result of the value expression: NEW ST_SpiralCurve(areferencelocation, alength, astartcurvature, anendcurvature, aspiraltype, NULL, NULL, aunit, ansrid).

22) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*,

f) a DOUBLE PRECISION value *astartm,*

g) a DOUBLE PRECISION value *anendm*,

h) an CHARACTER VARYING value *aunit.*

23) The null-call type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_SpiralCurve(areferencelocation, alength, astartcurvature, anendcurvature, aspiraltype, astartm, anendm, aunit, 0)*.

24) The method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)* takes the following input parameters:

a) an *ST_AffinePlacement* value *areferencelocation*,

b) a DOUBLE PRECISION value *alength,*

c) a DOUBLE PRECISION value *astartcurvature,*

d) a DOUBLE PRECISION value *anendcurvature,*

e) a CHARACTER VARYING value *aspiraltype*,

f) a DOUBLE PRECISION value *astartm,*

g) a DOUBLE PRECISION value *anendm*,

h) a CHARACTER VARYING value *aunit,*

i) an INTEGER value *ansrid.*

25) For the value *alength:*

a) The value for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

26) For the type-preserving SQL-invoked constructor method *ST_SpiralCurve(ST_AffinePlacement, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, DOUBLE PRECISION, DOUBLE PRECISION, CHARACTER VARYING, INTEGER)*:

Case:

a) If *areferencelocation* is the null value or if *alength* is the null value or if *astartcurvature* is the null value or if *anendcurvature* is the null value or if *aspiraltype* is the null value or if *aunit* is the null value or if *ansrid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_SpiralCurve* value with:

i) The *ST_PrivateDimension* attribute set to 1 (one).

ii) Case:

1) If *astartm* is NULL and *anendm* is NOT NULL or if *astartm* is NOT NULL and *anendm* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – missing measure value(s)*.

2) If *astartm* and *anendm* are both NOT NULL, then the *ST_PrivateIsMeasured* attribute set to 1 (one).

3) Otherwise, the *ST_PrivateIsMeasured* attribute set to 0 (zero).

iii) The *ST_PrivateCoordinateDimension* attribute set to *areferencelocation.ST_Location.ST_CoordDim* + *ST_PrivateIsMeasured*.

iv) The *ST_PrivateIs3D* attribute set to *areferencelocation.ST_Location.ST_Is3D.*

v) Case:

   1) If CARDINALITY(*areferencelocation.ST_RefDirections()*) not equal to 2, then an exception is raised: *SQL/MM Spatial exception – incorrect number of vectors*.

   2) Otherwise, the *ST_PrivateReferenceLocation* attribute set to *areferencelocation*.

vi) The *ST_PrivateLength* attribute set to *alength*.

vii) The *ST_PrivateStartCurvature* attribute set to *astartcurvature*.

viii) The *ST_PrivateEndCurvature* attribute set to *anendcurvature*.

ix) The *ST_PrivateSpiralType* attribute set to *aspiraltype*.

x) The *ST_PrivateStartM* attribute set to *astartm*.

xi) The *ST_PrivateEndM* attribute set to *anendm*.

xii) The spatial reference system identifier set to *ansrid*.

7.9.3 ST_RefLocation Methods

Purpose

Observe and mutate the attribute ST_PrivateReferenceLocation of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_RefLocation()
   RETURNS ST_AffinePlacement
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateReferenceLocation
      END

CREATE METHOD ST_RefLocation
   (areflocation ST_AffinePlacement)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      -- If areflocation is the null value, then raise an exception
      IF areflocation IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      -- Check if curve and affine placement location point are both 2D
      -- or both 3D
      IF SELF.ST_Is3D() <> areflocation.ST_Location().ST_Is3D() THEN
         SIGNAL SQLSTATE '2FF96'
            SET MESSAGE_TEXT = 'mixed Is3D';
      END IF;
      -- Check if affine placement reference directions has two vectors
      IF CARDINALITY(areflocation.ST_RefDirections()) <> 2 THEN
         SIGNAL SQLSTATE '2FF86'
            SET MESSAGE_TEXT = 'incorrect number of vectors';
      END IF;
      RETURN
         SELF.ST_PrivateReferenceLocation(areflocation)
   END
```

Description

1) The method *ST_RefLocation()* has no input parameters.

2) For the null-call method *ST_RefLocation()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateReferenceLocation* of SELF.

3) The method *ST_RefLocation(ST_AffinePlacement)* takes the following input parameters:

   a) an *ST_AffinePlacement* value *areflocation*.

4) For the type-preserving method *ST_RefLocation(ST_AffinePlacement)*:

   Case:

a) If *areflocation* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) If SELF and the *ST_PrivateLocation* attribute *ST_Point* value of *areflocation* are not either both 2D or both 3D, then an exception condition is raised: *SQL/MM Spatial exception – mixed Is3D*.

d) If the number of *ST_Vectors* in the *ST_PrivateReferenceDirections* attribute of *areflocation* is not equal to 2, then an exception condition is raised: *SQL/MM Spatial exception – incorrect number of vectors.*

e) Otherwise, return an *ST_SpiralCurve* value with the attribute *ST_PrivateReferenceLocation* set to *areflocation*.

7.9.4 ST_Length Methods

Purpose

Observe and mutate the attribute ST_PrivateLength of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_Length()
   RETURNS DOUBLE PRECISION
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateLength
      END

CREATE METHOD ST_Length
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateLength
      END

CREATE METHOD ST_Length
   (alength DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      -- If alength is the null value, then raise an exception
      IF alength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      RETURN
         SELF.ST_PrivateLength(alength);
   END
```

```
CREATE METHOD ST_Length
   (alength DOUBLE PRECISION,
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      -- If alength is the null value, then raise an exception
      IF alength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      RETURN
         SELF.ST_PrivateLength(alength);
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Length()* has no input parameters.
2) For the null-call method *ST_Length()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateLength* of SELF.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Length()* is in the linear unit of measure identified by <linear unit>.
      ii) Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.
3) The method *ST_Length(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_Length(CHARACTER VARYING)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateLength* of SELF.
5) For the method *ST_Length(CHARACTER VARYING)*:
   a) The value for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
6) The method *ST_Length(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *alength*.

7) For the type-preserving method *ST_Length(DOUBLE PRECISION)*:

   Case:

   a) If *alength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_SpiralCurve* value with the attribute *ST_PrivateLength* set to *alength*.

      Case:

      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Length()* is in the linear unit of measure identified by <linear unit>.

      ii) Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.

8) The method *ST_Length(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:

   a) a DOUBLE PRECISION value *alength*.

   b) a CHARACTER VARYING value *aunit*.

9) For the type-preserving method *ST_Length(DOUBLE PRECISION, CHARACTER VARYING)*:

   Case:

   a) If *alength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_SpiralCurve* value with the attribute *ST_PrivateLength* set to *alength*.

10) For the method *ST_Length(DOUBLE PRECISION, CHARACTER VARYING)*:

   a) The value for *aunit* shall be a supported <unit name>.

   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

   c) If the unit specified by *aunit* is not supported by the implementation to compute the length of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

7.9.5 ST_StartCurvature Methods

Purpose

Observe and mutate the attribute ST_PrivateStartCurvature of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_StartCurvature()
   RETURNS DOUBLE PRECISION
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartCurvature
      END

CREATE METHOD ST_StartCurvature
   (astartcurvature DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      -- If astartcurvature is the null value, then raise an exception
      IF astartcurvature IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      RETURN
         SELF.ST_PrivateStartCurvature(astartcurvature);
   END
```

Description

1) The method *ST_StartCurvature()* has no input parameters.
2) For the null-call method *ST_StartCurvature()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the attribute *ST_PrivateStartCurvature* of SELF.
3) The method *ST_StartCurvature(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *astartcurvature*.
4) For the type-preserving method *ST_StartCurvature(DOUBLE PRECISION)*:

   Case:

   a) If *astartcurvature* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_SpiralCurve* value with the attribute *ST_PrivateStartCurvature* set to *astartcurvature*.

7.9.6 ST_EndCurvature Methods

Purpose

Observe and mutate the attribute ST_PrivateEndCurvature of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_EndCurvature()
   RETURNS DOUBLE PRECISION
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndCurvature
      END

CREATE METHOD ST_EndCurvature
   (anendcurvature DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      -- If anendcurvature is the null value, then raise an exception
      IF anendcurvature IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      RETURN
         SELF.ST_PrivateEndCurvature(anendcurvature);
   END
```

Description

1) The method *ST_EndCurvature()* has no input parameters.

2) For the null-call method *ST_EndCurvature()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateEndCurvature* of SELF.

3) The method *ST_EndCurvature(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *anendcurvature*.

4) For the type-preserving method *ST_EndCurvature(DOUBLE PRECISION)*:

   Case:

   a) If *anendcurvature* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_SpiralCurve* value with the attribute *ST_PrivateEndCurvature* set to *anendcurvature*.

7.9.7 ST_SpiralType Methods

Purpose

Observe and mutate the attribute ST_PrivateSpiralType of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_SpiralType()
   RETURNS CHARACTER VARYING(64)
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateSpiralType
      END

CREATE METHOD ST_SpiralType
   (aspiraltype CHARACTER VARYING(64))
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      -- If aspiraltype is the null value, then raise an exception
      IF aspiraltype IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      RETURN
         SELF.ST_PrivateSpiralType(aspiraltype);
   END
```

Description

1) The method *ST_SpiralType()* has no input parameters.

2) For the null-call method *ST_SpiralType()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateSpiralType* of SELF.

3) The method *ST_SpiralType(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *aspiraltype*.

4) For the type-preserving method *ST_SpiralType(CHARACTER VARYING)*:

   Case:

   a) If *aspiraltype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_SpiralCurve* value with the attribute *ST_PrivateSpiralType* set to *aspiraltype*.

7.9.8 ST_StartM Methods

Purpose

Observe and mutate the attribute ST_PrivateStartM of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_StartM()
   RETURNS DOUBLE PRECISION
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateStartM
      END

CREATE METHOD ST_StartM
   (astartm DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateEndM is NOT NULL THEN
         SET measured = 1;
      -- If astartm is NULL, IS_Measured must be 0 (zero)
      IF astartm IS NULLTHEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateStartM(astartm);
   END
```

Description

1) The method *ST_StartM()* has no input parameters.

2) For the null-call method *ST_StartM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateStartM* of SELF.

3) The method *ST_StartM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *astartm*.

4) For the type-preserving method *ST_StartM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0 (zero).

      ii) If *SELF.ST_PrivateEndM* is NOT NULL, set *M* = 1 (one).

      iii) If *astartm* IS NULL, set *M* to 0 (zero).

iv) Return an *ST_SpiralCurve* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_StartM* attribute set to *astartm*.

7.9.9 ST_EndM Methods

Purpose

Observe and mutate the attribute ST_PrivateEndM of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_EndM()
   RETURNS DOUBLE PRECISION
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateEndM
      END

CREATE METHOD ST_EndM
   (anendm DOUBLE PRECISION)
   RETURNS ST_SpiralCurve
   FOR ST_SpiralCurve
   BEGIN
      DECLARE measured INTEGER;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_SpiralCurve);
      END IF;
      SET measured = 0;
      IF SELF.ST_PrivateStartM is NOT NULL THEN
         SET measured = 1;
      -- If anendm is NULL, IS_Measured must be 0 (zero)
      IF anendm IS NULL THEN
         SET measured = 0;
      RETURN
         SELF.ST_PrivateIsMeasured(measured).
            ST_PrivateEndM(anendm);
   END
```

Description

1) The method *ST_EndM()* has no input parameters.

2) For the null-call method *ST_EndM()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the attribute *ST_PrivateEndM* of SELF.

3) The method *ST_EndM(DOUBLE PRECISION)* takes the following input parameters:

   a) an *DOUBLE PRECISION* value *anendm*.

4) For the type-preserving method *ST_EndM(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise:

      i) Let *M* be 0.

      ii) If *SELF.ST_PrivateStartM* is NOT NULL, set *M* = 1 (one).

      iii) If *anendm* IS NULL,set *M* to 0 (zero).

iv) Return an *ST_SpiralCurve* value with:

1) The *ST_PrivateIsMeasured* attribute set to *M*.

2) The *ST_EndM* attribute set to *anendm*.

7.9.10 ST_StartPoint Method

Purpose

Return the start point of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.

2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Let *SP* be the *ST_Point* start point value calculated from the *ST_PrivateReferenceLocation, ST_PrivateLength, ST_PrivateStartCurvature, ST_PrivateEndCurvature,* and *ST_PrivateSpiralType* attribute values of SELF.

      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *SP = SP.ST_M(SELF.ST_StartM())*.

      iii) Return *SP*.

7.9.11 ST_EndPoint Method

Purpose

Return the end point of an ST_SpiralCurve value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_SpiralCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            BEGIN
               --
               -- See Description
               --
            END;
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.
2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Let *EP* be the *ST_Point* end point value calculated from the *ST_PrivateReferenceLocation, ST_PrivateLength, ST_PrivateStartCurvature, ST_PrivateEndCurvature,* and *ST_PrivateSpiralType* attribute values of SELF.
      ii) If *SELF.ST_Is_Measured()* is equal to 1 (one), then *EP* = *EP.ST_M(SELF.ST_EndM())*.
      iii) Return *EP*.

7.9.12 ST_SpiralFromTxt Functions

Purpose

Return an ST_SpiralCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_SpiralCurve value.

Definition

```
CREATE FUNCTION ST_SpiralFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_SpiralFromTxt(awkt, 0)

CREATE FUNCTION ST_SpiralFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_SpiralFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_SpiralFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_SpiralFromTxt(awkt, 0)*.
3) The function *ST_SpiralFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_SpiralFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_SpiralCurve* value.

      If *awkt* is not producible in the BNF for <spiral text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_SpiralCurve)*.

7.9.13 ST_SpiralFromWKB Functions

Purpose

Return an ST_SpiralCurve value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_SpiralCurve value.

Definition

```
CREATE FUNCTION ST_SpiralFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_SpiralFromWKB(awkb, 0)

CREATE FUNCTION ST_SpiralFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_SpiralFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_SpiralFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_SpiralFromWKB(awkb, 0)*.
3) The function *ST_SpiralFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_SpiralFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkb* is the well-known binary representation of an *ST_SpiralCurve* value.

      If *awkb* is not producible in the BNF for <spiral binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_SpiralCurve)*.

7.9.14 ST_SpiralFromGML Functions

Purpose

Return an ST_SpiralCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML SpiralCurve representation of an ST_SpiralCurve value.

Definition

```
CREATE FUNCTION ST_SpiralFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_SpiralFromGML(agml, 0)

CREATE FUNCTION ST_SpiralFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_SpiralCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_SpiralFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_SpiralFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_SpiralFromGML(agml, 0)*.
3) The function *ST_SpiralFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_SpiralFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a SpiralCurve XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_SpiralCurve)*.

7.10 ST_CompoundCurve Type and Routines

7.10.1 ST_CompoundCurve Type

Purpose

The general notion of a compound curve is a sequence of contiguous curves such that adjacent curves are joined at their end points. The contributing curve types include all subtypes of ST_Curve. Furthermore, the end point of each curve shall be coincident with the start point of the next curve in the list.

Definition

```
CREATE TYPE ST_CompoundCurve
   UNDER ST_Curve
   AS (
      ST_PrivateCurves ST_Curve
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_CompoundCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_CompoundCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CompoundCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_CompoundCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CompoundCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_CompoundCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CompoundCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_CompoundCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_CompoundCurve(acurve ST_Curve)
   RETURNS ST_CompoundCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CompoundCurve
   (acurve ST_Curve,
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CompoundCurve
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CompoundCurve
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Curves()
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Curves
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumCurves()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_CurveN
   (aposition INTEGER)
   RETURNS ST_Curve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_StartPoint()
   RETURNS ST_Point,

OVERRIDING METHOD ST_EndPoint()
   RETURNS ST_Point
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivateCurves* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateCurves*.

Description

1) The *ST_CompoundCurve* type provides for public use:
   a) a method *ST_CompoundCurve(CHARACTER LARGE OBJECT)*,
   b) a method *ST_CompoundCurve(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_CompoundCurve(BINARY LARGE OBJECT)*,
   d) a method *ST_CompoundCurve(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_CompoundCurve(ST_Curve)*,
   f) a method *ST_CompoundCurve(ST_Curve, INTEGER)*,
   g) a method *ST_CompoundCurve(ST_Curve ARRAY)*,
   h) a method *ST_CompoundCurve(ST_Curve ARRAY, INTEGER)*,
   i) a method *ST_Curves()*,
   j) a method *ST_Curves(ST_Curve ARRAY)*,
   k) a method *ST_NumCurves()*,
   l) a method *ST_CurveN(INTEGER)*,
   m) an overriding method *ST_StartPoint()*,
   n) an overriding method *ST_EndPoint()*,
   o) a function *ST_CompoundFromTxt(CHARACTER LARGE OBJECT)*,
   p) a function *ST_CompoundFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,
   q) a function *ST_CompoundFromWKB(BINARY LARGE OBJECT)*,
   r) a function *ST_CompoundFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   s) a function *ST_CompoundFromGML(CHARACTER LARGE OBJECT)*,
   t) a function *ST_CompoundFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The *ST_PrivateCurves* attribute contains a collection of *ST_Curve* values.

3) If each *ST_Curve* value in the *ST_PrivateCurves* attribute is well formed, then the *ST_CompoundCurve* value is well formed.
4) All the *ST_Curve* values in the *ST_PrivateCurves* attribute are in the same spatial reference system as the *ST_CompoundCurve* value.
5) The *ST_PrivateCurves* attribute shall not be the null value. The elements in the *ST_PrivateCurves* attribute shall not be the null value.
6) The coordinate dimension of an *ST_CompoundCurve* value is equal to the coordinate dimension of its *ST_Curve* values.
7) An *ST_CompoundCurve* value consists of one or more curves connected end to end. The contributing curve types include all subtypes of *ST_Curve* . Furthermore, the end point of each curve shall be coincident with the start point of the next curve in the list.
8) If an *ST_CompoundCurve* value is simple and closed, then it is considered a ring.
9) An *ST_CompoundCurve* value returned by the constructor function corresponds to the empty set.
10) An *ST_CompoundCurve* value with the cardinality of the attribute *ST_PrivateCurves* equal to 0 (zero) corresponds to the empty set.

7.10.2 ST_CompoundCurve Methods

Purpose

Return an ST_CompoundCurve value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Curve values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN NEW ST_CompoundCurve(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN NEW ST_CompoundCurve(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN ST_CompoundFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (acurve ST_Curve)
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN SELF.ST_SRID(0).ST_Curves(ARRAY[acurve])

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (acurve ST_Curve,
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN SELF.ST_SRID(ansrid).ST_Curves(ARRAY[acurve])

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN SELF.ST_SRID(0).ST_Curves(acurvearray)

CREATE CONSTRUCTOR METHOD ST_CompoundCurve
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   RETURN SELF.ST_SRID(ansrid).ST_Curves(acurvearray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_CompoundCurve(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_CompoundCurve(awktorgml, 0)*.
3) The method *ST_CompoundCurve(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(CHARACTER LARGE OBJECT, INTEGER)*:

   Case;

   a) If *awktorgml* contains a CompositeCurve XML element in the GML representation, then return the result of the value expression: *ST_CompoundFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_CompoundFromTxt(awktorgml, ansrid)*.
5) The method *ST_CompoundCurve(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_CompoundCurve(awkb, 0)*.
7) The method *ST_CompoundCurve(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
8) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_CompoundFromWKB(awkb, ansrid)*.
9) The method *ST_CompoundCurve(ST_Curve)* takes the following input parameters:
   a) an *ST_Curve* value *acurve*.
10) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(ST_Curve)* returns the result of the value expression: *NEW ST_CompoundCurve(acurve, 0)*.
11) The method *ST_CompoundCurve(ST_Curve, INTEGER)* takes the following input parameters:
    a) an *ST_Curve* value *acurve*,
    b) an INTEGER value *ansrid*.
12) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(ST_Curve, INTEGER)* returns the result of the value expression: *NEW ST_CompoundCurve(ARRAY[acurve], ansrid).*

13) The method *ST_CompoundCurve(ST_Curve ARRAY)* takes the following input parameters:

    a) an *ST_Curve* ARRAY value *acurvearray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(ST_Curve ARRAY)* returns the result of the value expression: *NEW ST_CompoundCurve(acurvearray, 0).*

15) The method *ST_CompoundCurve(ST_Curve ARRAY, INTEGER)* takes the following input parameters:

    a) an *ST_Curve* ARRAY value *acurvearray*,

    b) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_CompoundCurve(ST_Curve ARRAY, INTEGER)* returns an *ST_CompoundCurve* value with:

    a) The spatial reference system identifier set to *ansrid*.

    b) Using the method *ST_Curves(ST_Curve ARRAY)*:

        i) the *ST_PrivateDimension* attribute set to 1 (one).

        ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(acurvearray).*

        iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(acurvearray)*.

        iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(acurvearray)*.

        v) the *ST_PrivateCurves* attribute set to *acurvearray*.

7.10.3 ST_Curves Methods

Purpose

Observe and mutate the ST_PrivateCurves attribute of an ST_CompoundCurve value.

Definition

```
CREATE METHOD ST_Curves()
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_CompoundCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateCurves
      END

CREATE METHOD ST_Curves
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundCurve
   FOR ST_CompoundCurve
   BEGIN
      DECLARE counter INTEGER;

      -- If acurvearray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(acurvearray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_CompoundCurve);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and acurvearray.
      IF (CARDINALITY(acurvearray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(acurvearray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- If the start point of any curve is not coincident with the end
      -- point of the previous curve, then raise an exception
      SET counter = 2;
      WHILE counter <= CARDINALITY(acurvearray) DO
            IF acurvearray[counter].ST_StartPoint() <>
               acurvearray[counter-1].ST_EndPoint() THEN
            SIGNAL SQLSTATE '2FF11'
               SET MESSAGE_TEXT = 'non-contiguous curves';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      -- If SELF is the null value, then return the null value. Otherwise,
      -- return an ST_CompoundCurve value with the ST_PrivateCurves
      -- attribute set to acurvearray.
      RETURN
         SELF.ST_PrivateDimension(1).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(acurvearray)).
            ST_PrivateIs3D(ST_GetIs3D(acurvearray)).
            ST_PrivateIsMeasured(ST_GetIsMeasured(acurvearray)).
            ST_PrivateCurves(acurvearray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Curves()* has no input parameters.
2) For the null-call method *ST_Curves()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateCurves* attribute of SELF.
3) The method *ST_Curves(ST_Curve ARRAY)* takes the following input parameters:
   a) an *ST_Curve* ARRAY value *acurvearray*.
4) For the type-preserving method *ST_Curves(ST_Curve ARRAY)*:
   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *acurvearray* is the null value or contains null elements.
   b) Case:
      i) If SELF is the null value, then return the null value.
      ii) If the cardinality of *acurvearray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID(acurvearray)*, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.
      iii) If the start point of any *ST_Curve* value in *acurvearray* is not equal to the end point of the previous *ST_Curve* value in *acurvearray*, then an exception condition is raised: *SQL/MM Spatial exception – non-contiguous curves*.
      iv) Otherwise, return an *ST_CompoundCurve* value with:
         1) The dimension set to 1 (one).
         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(acurvearray)*.
         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(acurvearray)*.
         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(acurvearray)*.
         5) The *ST_PrivateCurves* attribute set to *acurvearray*.

7.10.4 ST_NumCurves Method

Purpose

Return the cardinality of the ST_PrivateCurves attribute of an ST_CompoundCurve value.

Definition

```
CREATE METHOD ST_NumCurves()
   RETURNS INTEGER
   FOR ST_CompoundCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivateCurves)
      END
```

Description

1) The method *ST_NumCurves()* has no input parameters.
2) For the null-call method *ST_NumCurves()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the cardinality of the *ST_PrivateCurves* attribute.

7.10.5 ST_CurveN Method

Purpose

Return the specified element in the ST_PrivateCurves attribute of an ST_CompoundCurve value.

Definition

```
CREATE METHOD ST_CurveN
   (aposition INTEGER)
   RETURNS ST_Curve
   FOR ST_CompoundCurve
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Curve);
      END IF;
      IF aposition < 1 OR
         aposition > CARDINALITY(SELF.ST_PrivateCurves) THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Curve);
         END;
      END IF;
      RETURN SELF.ST_PrivateCurves[aposition];
   END
```

Description

1) The method *ST_CurveN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_CurveN(INTEGER)*:

   Case:
   a) If SELF is an empty set, then return the null value.
   b) If *aposition* is less than 1 (one) or greater than the cardinality of the *ST_PrivateCurves* attribute, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return an *ST_Curve* value at element *aposition* in the *ST_PrivateCurves* attribute of SELF.

7.10.6 ST_StartPoint Method

Purpose

Return an ST_Point value that is the start point of an ST_CompoundCurve value.

Definition

```
CREATE METHOD ST_StartPoint()
   RETURNS ST_Point
   FOR ST_CompoundCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Curves()[1].ST_StartPoint()
      END
```

Description

1) The method *ST_StartPoint()* has no input parameters.
2) For the null-call method *ST_StartPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_Curves()[1].ST_StartPoint()*.

7.10.7 ST_EndPoint Method

Purpose

Return an ST_Point value that is the end point of an ST_CompoundCurve value.

Definition

```
CREATE METHOD ST_EndPoint()
   RETURNS ST_Point
   FOR ST_CompoundCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
               SELF.ST_Curves()[SELF.ST_NumCurves()].ST_EndPoint()
      END
```

Description

1) The method *ST_EndPoint()* has no input parameters.

2) For the null-call method *ST_EndPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the result of the value expression: *SELF.ST_Curves()[SELF.ST_NumCurves()].ST_EndPoint()*.

7.10.8 ST_CompoundFromTxt Functions

Purpose

Return an ST_CompoundCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CompoundCurve value.

Definition

```
CREATE FUNCTION ST_CompoundFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CompoundFromTxt(awkt, 0)

CREATE FUNCTION ST_CompoundFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_CompoundFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_CompoundFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CompoundFromTxt(awkt, 0)*.
3) The function *ST_CompoundFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CompoundFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_CompoundCurve* value.

      If *awkt* is not producible in the BNF for <compoundcurve text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_CompoundCurve)*.

7.10.9 ST_CompoundFromWKB Functions

Purpose

Return an ST_CompoundCurve value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CompoundCurve value.

Definition

```
CREATE FUNCTION ST_CompoundFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CompoundFromWKB(awkb, 0)

CREATE FUNCTION ST_CompoundFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_CompoundFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_CompoundFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_CompoundFromWKB(awkb, 0)*.
3) The function *ST_CompoundFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CompoundFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkb* is the well-known binary representation of an *ST_CompoundCurve* value.

      If *awkb* is not producible in the BNF for <compoundcurve binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_CompoundCurve)*.

7.10.10 ST_CompoundFromGML Functions

Purpose

Return an ST_CompoundCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_CompoundCurve value.

Definition

```
CREATE FUNCTION ST_CompoundFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CompoundFromGML(agml, 0)

CREATE FUNCTION ST_CompoundFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_CompoundCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_CompoundFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_CompoundFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CompoundFromGML(agml, 0)*.
3) The function *ST_CompoundFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CompoundFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a CompositeCurve XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_CompoundCurve)*.

8 Surface Types

8.1 ST_Surface Type and Routines

8.1.1 ST_Surface Type

Purpose

The ST_Surface type is a supertype for 2-dimensional geometry types.

Definition

```
CREATE TYPE ST_Surface
   UNDER ST_Geometry
   NOT INSTANTIABLE
   NOT FINAL

   METHOD ST_Area()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Area
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DArea()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DArea
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Perimeter()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Perimeter
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_3DPerimeter()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DPerimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Centroid()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DCentroid()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointOnSurface()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DPointOnSurf()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsWorld()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Is3DClosed()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_IsShell()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The *ST_Surface* type provides for public use:
   a) a method *ST_Area()*,
   b) a method *ST_Area(CHARACTER VARYING)*,
   c) a method *ST_3DArea()*,
   d) a method *ST_3DArea(CHARACTER VARYING)*,
   e) a method *ST_Perimeter()*,
   f) a method *ST_Perimeter(CHARACTER VARYING)*,
   g) a method *ST_3DPerimeter()*,
   h) a method *ST_3DPerimeter(CHARACTER VARYING)*,
   i) a method *ST_Centroid()*,
   j) a method *ST_3DCentroid()*,
   k) a method *ST_PointOnSurface()*,
   l) a method *ST_3DPointOnSurf()*,
   m) a method *ST_IsWorld(),*
   n) a method *ST_Is3DClosed(),*
   o) a method *ST_IsShell().*
2) An *ST_Surface* value is a 2-dimensional *ST_Geometry* value that consists of a single connected interior that is associated with one exterior ring and zero or more interior rings. *ST_Surface* values in three-dimensional coordinate space are isomorphic to planar *ST_Surface* values. Stitching together simple surfaces along their boundaries forms polyhedral *ST_Surface* values and polyhedral surfaces in three-dimensional coordinate space may not be planar.
3) The dimension of an *ST_Surface* value is 2.
4) An *ST_Surface* value is closed if it is isomorphic to the surface of a sphere, or some torus. For an *ST_Surface* value to be closed, *SELF.ST_Is3D()* must equal to 1 (one), and the z coordinate values are considered in the calculation of *ST_3DIsClosed*. If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.
5) The boundary of an *ST_Surface* value that is not closed is the collection of the exterior ring and interior rings of the *ST_Surface* value. The boundary of a closed *ST_Surface* value is the empty set.
6) If an *ST_Surface* value is simple and 3D closed, then it is called a shell.

8.1.2 ST_Area Methods

Purpose

Return the area measurement of an ST_Surface value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Area()
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_Area
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Area()* has no input parameters.
2) For the null-call method *ST_Area()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined area of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Area()* is in the linear unit of measure identified by <linear unit> squared.
      ii) Otherwise, the value returned by *ST_Area()* is in an implementation-defined unit of measure.
3) The method *ST_Area(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_Area(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the area of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
   d) Case:
      i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the implementation-defined area of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

8.1.3 ST_3DArea Methods

Purpose

Return the area measurement of an ST_Surface value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DArea()
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_3DArea
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DArea()* has no input parameters.
2) For the null-call method *ST_3DArea()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined area of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Area()* is in the linear unit of measure identified by <linear unit> squared.
      ii) Otherwise, the value returned by *ST_3DArea()* is in an implementation-defined unit of measure.
3) The method *ST_3DArea(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_3DArea(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the area of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) Otherwise, return the implementation-defined area of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

8.1.4 ST_Perimeter Methods

Purpose

Return the length measurement of the boundary of an ST_Surface value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Perimeter()
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Boundary().ST_Length()
      END

CREATE METHOD ST_Perimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_Boundary().ST_Length(aunit)
      END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Perimeter()* has no input parameters.
2) For the null-call method *ST_Perimeter()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined length of the boundary of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Perimeter()* is in the linear unit of measure identified by <linear unit> squared.
      ii) Otherwise, the value returned by *ST_Perimeter()* is in an implementation-defined unit of measure.
3) The method *ST_Perimeter(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_Perimeter(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the length of the boundary of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) Otherwise, return the implementation-defined length of the boundary of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

8.1.5 ST_3DPerimeter Methods

Purpose

Return the length measurement of the boundary of an ST_Surface value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DPerimeter()
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_3DBoundary().ST_3DLength()
      END

CREATE METHOD ST_3DPerimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_3DBoundary().ST_3DLength(aunit)
      END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DPerimeter()* has no input parameters.
2) For the null-call method *ST_3DPerimeter()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined length of the boundary of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DPerimeter()* is in the linear unit of measure identified by <linear unit> squared.
      ii) Otherwise, the value returned by *ST_3DPerimeter()* is in an implementation-defined unit of measure.
3) The method *ST_3DPerimeter(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_3DPerimeter(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the length of the boundary of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) Otherwise, return the implementation-defined length of the boundary of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

8.1.6 ST_Centroid Method

Purpose

Return the 2D ST_Point value that is the mathematical centroid of the ST_Surface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Centroid()
   RETURNS ST_Point
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
      -- ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_Centroid()* has no input parameters.
2) For the null-call method *ST_Centroid()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Return the mathematical centroid of the *ST_Surface* value. The result is not guaranteed to spatially intersect the *ST_Surface* value.
      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:
         1) The z coordinate values are not considered in the calculation.
         2) The *ST_Point* value does not include the z coordinate value.
      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
         1) The m coordinate values are not considered in the calculation.
         2) The *ST_Point* value does not include the m coordinate value.
      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

8.1.7 ST_3DCentroid Method

Purpose

Return the ST_Point value that is the mathematical centroid of the ST_Surface value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DCentroid()
   RETURNS ST_Point
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
      -- ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_3DCentroid()* has no input parameters.

2) For the null-call method *ST_3DCentroid()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Return the mathematical centroid of the *ST_Surface* value. The result is not guaranteed to spatially intersect the *ST_Surface* value.

      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:

         1) The z coordinate values are considered in the calculation.

         2) The *ST_Point* value includes the z coordinate value.

      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:

         1) The m coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the m coordinate value.

      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

8.1.8 ST_PointOnSurface Method

Purpose

Return an ST_Point value guaranteed to spatially intersect the ST_Surface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_PointOnSurface()
   RETURNS ST_Point
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
      -- ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_PointOnSurface()* has no input parameters.
2) For the null-call method *ST_PointOnSurface()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Return an *ST_Point* value guaranteed to spatially intersect the *ST_Surface* value.
      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:
         1) The z coordinate values are not considered in the calculation.
         2) The *ST_Point* value does not include the z coordinate value.
      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
         1) The m coordinate values are not considered in the calculation.
         2) The *ST_Point* value does not include the m coordinate value.
      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

8.1.9 ST_3DPointOnSurf Method

Purpose

Return an ST_Point value guaranteed to spatially intersect the ST_Surface value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DPointOnSurf()
   RETURNS ST_Point
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
      -- ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_3DPointOnSurf()* has no input parameters.

2) For the null-call method *ST_3DPointOnSurf()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Return an *ST_Point* value guaranteed to spatially 3D intersect the *ST_Surface* value.

      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:

         1) The z coordinate values are considered in the calculation.

         2) The *ST_Point* value includes the z coordinate value.

      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:

         1) The m coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the m coordinate value.

      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

8.1.10 ST_IsWorld Method

Purpose

Test if the exterior of the ST_Surface value is the empty set, ignoring z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_IsWorld()
   RETURNS INTEGER
   FOR ST_Surface
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsWorld()* has no input parameters.
2) For the null-call method *ST_IsWorld()*:

   Case:

   a) If the exterior of the *ST_Surface* value corresponds to the empty set, then return 1 (one).
   b) Otherwise, return 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

8.1.11 ST_3DIsClosed Method

Purpose

Test if an ST_Surface value is closed, considering z (but not m) coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DIsClosed()
   RETURNS INTEGER
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            0
         ELSE
            SELF.ST_3DBoundary().ST_IsEmpty()
      END
```

Description

1) The method *ST_3DIsClosed()* has no input parameters.
2) For the null-call method *ST_3DIsClosed()*:

   Case:

   a) If SELF is an empty set, then return 0 (zero).
   b) If the boundary of the *ST_Surface* value is the empty set, then 1 (one).
   c) Otherwise, 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

8.1.12 ST_IsShell Method

Purpose

Test if an ST_Surface value is a shell, considering z (but not m) coordinate values in the calculations.

Definition

```
CREATE METHOD ST_IsShell()
   RETURNS INTEGER
   FOR ST_Surface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            0
         WHEN (SELF.ST_3DIsSimple() = 1 AND SELF.ST_3DIsClosed() = 1) THEN
            1
         ELSE
            0
      END
```

Description

1) The method *ST_IsShell()* has no input parameters.
2) For the null-call method *ST_IsShell()*:

   Case:

   a) If SELF is an empty set, then return 0 (zero).
   b) If SELF is simple and SELF is closed, then return 1 (one).
   c) Otherwise 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

8.2 ST_CurvePolygon Type and Routines

8.2.1 ST_CurvePolygon Type

Purpose

The ST_CurvePolygon type is a subtype of the ST_Surface type and values represent a planar surface whose boundary is specified by one exterior ring and zero or more interior rings. Each interior ring defines a hole in the curve polygon.

Definition

```
CREATE TYPE ST_CurvePolygon
   UNDER ST_Surface
   AS (
      ST_PrivateExteriorRing ST_Curve,
      ST_PrivateInteriorRings ST_Curve
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_CurvePolygon
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_CurvePolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CurvePolygon
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_CurvePolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CurvePolygon
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_CurvePolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CurvePolygon
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_CurvePolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve)
   RETURNS ST_CurvePolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve,
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve,
    acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CurvePolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve,
    acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ExteriorRing()
   RETURNS ST_Curve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ExteriorRing(acurve ST_Curve)
   RETURNS ST_CurvePolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_InteriorRings()
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_InteriorRings
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CurvePolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumInteriorRing()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_InteriorRingN
   (aposition INTEGER)
   RETURNS ST_Curve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_CurvePolyToPoly()
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivateExteriorRing* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateExteriorRing*.
5) The attribute *ST_PrivateInteriorRings* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateInteriorRings*.

Description

1) The *ST_CurvePolygon* type provides for public use:
   a) a method *ST_CurvePolygon(CHARACTER LARGE OBJECT)*,
   b) a method *ST_CurvePolygon(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_CurvePolygon(BINARY LARGE OBJECT)*,
   d) a method *ST_CurvePolygon(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_CurvePolygon(ST_Curve)*,
   f) a method *ST_CurvePolygon(ST_Curve, INTEGER)*,
   g) a method *ST_CurvePolygon(ST_Curve, ST_Curve ARRAY)*,
   h) a method *ST_CurvePolygon(ST_Curve, ST_Curve ARRAY, INTEGER)*,

i) a method *ST_ExteriorRing()*,

j) a method *ST_ExteriorRing(ST_Curve)*,

k) a method *ST_InteriorRings()*,

l) a method *ST_InteriorRings(ST_Curve ARRAY)*,

m) a method *ST_NumInteriorRing()*,

n) a method *ST_InteriorRingN(INTEGER)*,

o) a method *ST_CurvePolyToPoly()*,

p) a function *ST_CPolyFromText(CHARACTER LARGE OBJECT)*,

q) a function *ST_CPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*,

r) a function *ST_CPolyFromWKB(BINARY LARGE OBJECT)*,

s) a function *ST_CPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*,

t) a function *ST_CPolyFromGML(CHARACTER LARGE OBJECT)*,

u) a function *ST_CPolyFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateExteriorRing* attribute is an *ST_Curve* value that is a ring.

3) The *ST_PrivateInteriorRings* attribute is a collection of *ST_Curve* values. Each *ST_Curve* value in the collection is a ring.

4) The *ST_PrivateExteriorRing* attribute shall not be the null value.

5) The *ST_PrivateInteriorRings* attribute shall not be the null value. The elements in the *ST_PrivateInteriorRings* attribute shall not be the null value. If the *ST_CurvePolygon* value does not have interior rings, then the *ST_PrivateInteriorRings* attribute is set to an empty *ST_Curve* ARRAY value.

6) All the *ST_Curve* values in the *ST_PrivateExteriorRing* attribute and *ST_PrivateInteriorRings* attribute shall be in the same spatial reference system as the *ST_CurvePolygon* value.

7) The coordinate dimension of an *ST_CurvePolygon* value is equal to the coordinate dimension of its *ST_Curve* values.

8) An *ST_CurvePolygon* value is simple.

9) The ring in the *ST_PrivateExteriorRing* attribute and the rings in the *ST_PrivateInteriorRings* attribute represent the boundary of the *ST_CurvePolygon* value.

10) An *ST_CurvePolygon* value is topologically closed.

11) The rings in the boundary may spatially intersect at most only a single point:

$$\forall\ p \in \mathit{ST\_CurvePolygon},\ \forall\ c_1, c_2 \in \text{Boundary}(p),\ c_1 \neq c_2,$$
$$\forall\ a_1, a_2 \in \mathit{ST\_Point},\ a_1, a_2 \in c_1,\ a_1 \neq a_2,\ [\ a_1 \in c_2 \Rightarrow a_2 \notin c_2\ ]$$

12) An *ST_CurvePolygon* value shall not have cut lines, spikes or punctures:

$$\forall\ p \in \mathit{ST\_CurvePolygon},\ p = \text{Closure}(\text{Interior}(p))$$

13) The interior of every *ST_CurvePolygon* value is a connected point set.

14) The exterior of an *ST_CurvePolygon* with one or more holes is not connected. Each hole defines a dis-connected component of the exterior.

15) An *ST_CurvePolygon* is a topologically closed point set.

16) An *ST_CurvePolygon* value returned by the constructor function corresponds to the empty set.

17) An *ST_CurvePolygon* value corresponds to the empty set if the *ST_PrivateExteriorRing* attribute corresponds to the empty set.

18) An *ST_CurvePolygon* value is well formed only if all the *ST_Curve* values in the *ST_PrivateExteriorRing* attribute and *ST_PrivateInteriorRings* attribute are well formed.

8.2.2 ST_CurvePolygon Methods

Purpose

Return an ST_CurvePolygon value constructed from either the well-known text representation, the well-known binary representation, a GML representation, or the specified ST_Curve values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN NEW ST_CurvePolygon(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN NEW ST_CurvePolygon(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN ST_CPolyFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve)
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN SELF.ST_SRID(0).ST_ExteriorRing(acurve).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_Curve ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve,
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorRing(acurve).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_Curve ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve,
    acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN SELF.ST_SRID(0).ST_ExteriorRing(acurve).
      ST_InteriorRings(acurvearray)
```

```
CREATE CONSTRUCTOR METHOD ST_CurvePolygon
   (acurve ST_Curve,
    acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorRing(acurve).
      ST_InteriorRings(acurvearray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_CurvePolygon(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_CurvePolygon(awktorgml, 0)*.

3) The method *ST_CurvePolygon(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Polygon or PolygonPatch XML element in the GML representation, then return the result of the value expression: *ST_CPolyFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_CPolygFromText(awktorgml, ansrid)*.

5) The method *ST_CurvePolygon(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_CurvePolygon(awkb, 0)*.

7) The method *ST_CurvePolygon(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_CPolygFromWKB(awktorgml, ansrid)*.

9) The method *ST_CurvePolygon(ST_Curve)* takes the following input parameters:

   b) an *ST_Curve* value *acurve*.

10) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(ST_Curve)* returns an *ST_CurvePolygon* value with:

   a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *acurve*, the *ST_PrivateCoordinateDimension* attribute set to *acurve*.*ST_PrivateCoordinateDimension*, the *ST_PrivateDimension* attribute set to 2, the *ST_PrivateIs3D* attribute set to *acurve.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured* attribute set to *acurve.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_Curve* ARRAY value.

11) The method *ST_CurvePolygon(ST_Curve, INTEGER)* takes the following input parameters:

a) an *ST_Curve* value *acurve*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(ST_Curve, INTEGER)* returns an *ST_CurvePolygon* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *acurve*, the *ST_PrivateCoordinateDimension* attribute set to *acurve*.*ST_PrivateCoordinateDimension*, the *ST_PrivateDimension* attribute set to 2, the *ST_PrivateIs3D* attribute set to *acurve.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured* attribute set to *acurve.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_Curve* ARRAY value.

13) The method *ST_CurvePolygon(ST_Curve, ST_Curve ARRAY)* takes the following input parameters:

a) an *ST_Curve* value *acurve*,

b) an *ST_Curve* ARRAY value *acurvearray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(ST_Curve, ST_Curve ARRAY)* returns an *ST_CurvePolygon* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *acurve*, the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(acurve, acurvearray)*, the *ST_PrivateDimension* attribute set to 2, the *ST_PrivateIs3D* attribute set to *acurve.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured* attribute set to *acurve.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to *acurvearray*.

15) The method *ST_CurvePolygon(ST_Curve, ST_Curve ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Curve* value *acurve*,

b) an *ST_Curve* ARRAY value *acurvearray*,

c) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_CurvePolygon(ST_Curve, ST_Curve ARRAY, INTEGER)* returns an *ST_CurvePolygon* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *acurve*, the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(acurve, acurvearray)*, the *ST_PrivateDimension* attribute set to 2, the *ST_PrivateIs3D* attribute set to *acurve.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured* attribute set to *acurve.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to *acurvearray*.

8.2.3 ST_ExteriorRing Methods

Purpose

Observe and mutate the ST_PrivateExteriorRing attribute of an ST_CurvePolygon value.

Definition

```
CREATE METHOD ST_ExteriorRing()
   RETURNS ST_Curve
   FOR ST_CurvePolygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateExteriorRing
      END

CREATE METHOD ST_ExteriorRing
   (acurve ST_Curve)
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   BEGIN
      DECLARE acounter INTEGER;

      IF acurve IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_CurvePolygon);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and acurve.
      IF SELF.ST_SRID() <> acurve.ST_SRID() THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- If acurve is not a ring, then raise an exception.
      IF acurve.ST_Is3D() = 0 and acurve.ST_IsRing() = 0 THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      IF acurve.ST_Is3D() = 1 and acurve.ST_3DIsRing() = 0 THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      -- For all interior rings
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(SELF.ST_InteriorRings()) DO
         -- If the current interior ring is not within
         -- acurve as a curve polygon, then raise an exception
         IF SELF.ST_InteriorRings()[acounter].ST_Within(
            SELF.ST_CurvePolygon(acurve, SELF.ST_SRID())) = 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         -- If the current interior ring intersects acurve
         -- with a dimension greater than 0 (zero), then
         -- raise an exception.
```

```
         IF SELF.ST_InteriorRings()[acounter].ST_Intersection(acurve).
            ST_Dimension() > 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         SET acounter = acounter + 1;
      END WHILE;
      -- Return an ST_CurvePolygon value with the ST_PrivateExteriorRing
      -- attribute set to acurve.
      RETURN
         SELF.ST_PrivateDimension(2).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(acurve,
               SELF.ST_PrivateInteriorRings)).
            ST_PrivateIs3D(acurve.ST_PrivateIs3D()).
            ST_PrivateIsMeasured(acurve.ST_PrivateIsMeasured()).
            ST_PrivateExteriorRing(acurve);
   END
```

Description

1) The method *ST_ExteriorRing()* has no input parameters.

2) For the null-call method *ST_ExteriorRing()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateExteriorRing* attribute of SELF.

3) The method *ST_ExteriorRing(ST_Curve)* takes the following input parameters:

   a) an *ST_Curve* value *acurve*.

4) For the type-preserving method *ST_ExteriorRing(ST_Curve)*:

   Case:

   a) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) If the spatial reference system of SELF is not equal to the spatial reference system of *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

   d) Case:

      i) If *acurve* is not 3D and *acurve* is not a ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

      ii) If *acurve* is 3D and *acurve* is not a 3D ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   e) If any two rings in *acurve* and the interior rings of SELF spatially intersect with dimension of the result greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   f) If any ring returned as an *ST_Curve* element of the *ST_Curve* ARRAY returned by *ST_InteriorRings()* is not spatially within an *ST_CurvePolygon* value formed from the exterior ring of SELF, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   g) Otherwise, return an *ST_CurvePolygon* value with:

      i) The dimension set to 2.

      ii) The coordinate dimension set to the value expression: *ST_GetCoordDim(acurve, SELF.ST_PrivateInteriorRings)*.

      iii) The *ST_PrivateIs3D* attribute set to the value expression: *acurve.ST_PrivateIs3D()*.

iv) The *ST_PrivateIsMeasured* attribute set to the value expression: *acurve.ST_PrivateIsMeasured()*.

v) The *ST_PrivateExteriorRing* attribute set to *acurve*.

8.2.4 ST_InteriorRings Methods

Purpose

Observe and mutate the ST_PrivateInteriorRings attribute of an ST_CurvePolygon value.

Definition

```
CREATE METHOD ST_InteriorRings()
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_CurvePolygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateInteriorRings
      END

CREATE METHOD ST_InteriorRings
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CurvePolygon
   FOR ST_CurvePolygon
   BEGIN
      DECLARE acounter INTEGER;
      DECLARE bcounter INTEGER;

      IF SELF.ST_ExteriorRing() IS NULL THEN
         SIGNAL SQLSTATE '2FF07'
            SET MESSAGE_TEXT = 'null exterior ring';
      END IF;
      -- If acurvearray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(acurvearray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN SELF;
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and acurvearray.
      IF (CARDINALITY(acurvearray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(acurvearray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- If any ST_Curve value is not a ring, then
      -- raise an exception.
      IF acurve.ST_Is3D() = 0 THEN
         SET acounter = 1;
         WHILE acounter <= CARDINALITY(acurvearray) DO
            IF acurvearray[acounter].ST_IsRing() = 0 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            SET acounter = acounter + 1;
         END WHILE;
      IF acurve.ST_Is3D() = 1 THEN
         SET acounter = 1;
         WHILE acounter <= CARDINALITY(acurvearray) DO
            IF acurvearray[acounter].ST_3DIsRing() = 0 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
```

```
            END IF;
            SET acounter = acounter + 1;
         END WHILE;
      -- For all rings in acurvearray
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(acurvearray) DO
         -- If the current interior ring not within
         -- the exterior ring as a curve polygon, then raise an exception
         IF acurvearray[acounter].ST_Within(
            SELF.ST_CurvePolygon(SELF.ST_ExteriorRing(),
            SELF.ST_SRID())) = 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         -- If the current interior ring intersects the exterior
         -- ring with a dimension greater than zero, then
         -- raise an exception.
         IF acurvearray[acounter].ST_Intersection(
            SELF.ST_ExteriorRing()).ST_Dimension() > 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         SET acounter = acounter + 1;
      END WHILE;
      SET acounter = 1;
      -- For each ring pair in acurvearray
      WHILE acounter <= CARDINALITY(acurvearray)-1 DO
         SET bcounter = acounter+1;
         WHILE bcounter <= CARDINALITY(acurvearray) DO
            -- If the current interior ring pair overlap, then
            -- raise an exception.
            IF SELF.ST_CurvePolygon(acurvearray[acounter],
               SELF.ST_SRID()).ST_Overlaps(
               SELF.ST_CurvePolygon(acurvearray[bcounter],
               SELF.ST_SRID())) = 1 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            -- If the current interior ring pair intersect
            -- with a dimension greater than zero, then
            -- raise an exception.
            IF acurvearray[acounter].ST_Intersection(
               acurvearray[bcounter]).ST_Dimension() > 0 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            SET bcounter = bcounter + 1;
         END WHILE;
         SET acounter = acounter + 1;
      END WHILE;
      -- Return an ST_CurvePolygon value with the ST_PrivateInteriorRings
      -- attribute set to acurvearray.
      RETURN SELF.ST_PrivateCoordinateDimension(ST_GetCoordDim(
            SELF.ST_PrivateExteriorRing, acurvearray)).
         ST_PrivateInteriorRings(acurvearray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_InteriorRings()* has no input parameters.
2) For the null-call method *ST_InteriorRings()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateInteriorRings* attribute of SELF.
3) The method *ST_InteriorRings(ST_Curve ARRAY)* takes the following input parameters:
   a) an *ST_Curve* ARRAY value *acurvearray*.
4) For the type-preserving method *ST_InteriorRings(ST_Curve ARRAY)*:

   Case:

   a) If *SELF.ST_ExteriorRing()* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null exterior ring*.
   b) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *acurvearray* is the null value or contains null elements.
   c) If SELF is the null value, then return the null value.
   d) If the cardinality of *acurvearray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID*(*acurvearray*), then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.
   e) Case:
      i) If any *ST_Curve* value in *acurvearray* is not 3D and not a ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
      ii) If any *ST_Curve* value in *acurvearray* is 3D and not a 3D ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   f) If any rings in *acurvearray* and the exterior ring of SELF spatially intersect with dimension of the result greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   g) If any ring in *acurvearray* is not spatially within an *ST_CurvePolygon* value formed from the exterior ring of SELF, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   h) If any two rings in *acurvearray*, formed into *ST_CurvePolygon* values with no interior rings spatially overlap, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   i) If the intersection of any two rings in *acurvearray* has a dimension greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   j) Otherwise, return an *ST_CurvePolygon* value with
      i) The coordinate dimension set to the value expression: *ST_GetCoordDim(SELF.ST_PrivateExteriorRing, acurvearray)*.
      ii) The *ST_PrivateInteriorRings* attribute set to *acurvearray*.

8.2.5 ST_NumInteriorRing Method

Purpose

Return the cardinality of the ST_PrivateInteriorRings attribute of an ST_CurvePolygon value.

Definition

```
CREATE METHOD ST_NumInteriorRing()
   RETURNS INTEGER
   FOR ST_CurvePolygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivateInteriorRings)
      END
```

Description

1) The method *ST_NumInteriorRing()* has no input parameters.
2) For the null-call method *ST_NumInteriorRing()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the cardinality of the *ST_PrivateInteriorRings* attribute.

8.2.6 ST_InteriorRingN Method

Purpose

Return the specified element in the ST_PrivateInteriorRings attribute of an ST_CurvePolygon value.

Definition

```
CREATE METHOD ST_InteriorRingN
   (aposition INTEGER)
   RETURNS ST_Curve
   FOR ST_CurvePolygon
   BEGIN
      IF SELF.ST_IsEmpty() = 1  THEN
         RETURN CAST (NULL AS ST_Curve);
      END IF;
      IF aposition < 1 OR
         aposition > CARDINALITY(SELF.ST_PrivateInteriorRings) THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Curve);
         END;
      END IF;
      RETURN SELF.ST_PrivateInteriorRings[aposition];
   END
```

Description

1) The method *ST_InteriorRingN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_InteriorRingN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If *aposition* is less than one or greater than the cardinality of the *ST_PrivateInteriorRings* attribute, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return an *ST_Curve* value at element *aposition* in the *ST_PrivateInteriorRings* attribute of SELF.

8.2.7 ST_CurvePolyToPoly Method

Purpose

Return the ST_Polygon approximation of an ST_CurvePolygon value, considering z and m coordinate values in the calculations and including z and m coordinate values in the resultant geometry.

Definition

```
CREATE METHOD ST_CurvePolyToPoly()
   RETURNS ST_Polygon
   FOR ST_CurvePolygon
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_CurvePolyToPoly()* has no input parameters.

2) For the null-call method *ST_CurvePolyToPoly()*:

   Case:

   a) If SELF is an empty set, then return an empty set of type *ST_Polygon*.

   b) Otherwise, return the implementation-defined *ST_Polygon* value approximation of the *ST_CurvePolygon* value.

3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation and are included in the resultant geometry.

4) If SELF.ST_IsMeasured() is equal to 1 (one), then m coordinate values are calculated for the ST_Curve.ST_PrivatePoints ST_Point values by linear interpolation based on curve length using an implementation-defined interpolation algorithm. The resultant m coordinate values are included in the resultant geometry.

8.2.8 ST_CPolyFromText Functions

Purpose

Return an ST_CurvePolygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CurvePolygon value.

Definition

```
CREATE FUNCTION ST_CPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CPolyFromText(awkt, 0)

CREATE FUNCTION ST_CPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_CPolyFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_CPolyFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CPolyFromText(awkt, 0)*.
3) The function *ST_CPolyFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_CurvePolygon* value.

      If *awkt* is not producible in the BNF for <curvepolygon text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_CurvePolygon)*.

8.2.9 ST_CPolyFromWKB Functions

Purpose

Return an ST_CurvePolygon value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CurvePolygon value.

Definition

```
CREATE FUNCTION ST_CPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CPolyFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_CPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_CPolyFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_CPolyFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_CPolyFromWKB(awkb, 0)*.
3) The function *ST_CPolyFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_CurvePolygon* value.

      If *awkb* is not producible in the BNF for <curvepolygon binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_CurvePolygon)*.

8.2.10 ST_CPolyFromGML Functions

Purpose

Return an ST_CurvePolygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Polygon or PolygonPatch representation of an ST_CurvePolygon value.

Definition

```
CREATE FUNCTION ST_CPolyFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CPolyFromGML(agml, 0)

CREATE FUNCTION ST_CPolyFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_CurvePolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_CPolyFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_CPolyFromGML*(CHARACTER LARGE OBJECT) returns the result of the value expression: *ST_CPolyFromGML(agml, 0)*.
3) The function *ST_CPolyFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CPolyFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a Polygon or PolygonPatch XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_CurvePolygon)*.

8.3 ST_Polygon Type and Routines

8.3.1 ST_Polygon Type

Purpose

The ST_Polygon type is a subtype of the ST_CurvePolygon type and represents a planar surface whose boundary is defined by linear rings.

Definition

```
CREATE TYPE ST_Polygon
   UNDER ST_CurvePolygon
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Polygon
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_Polygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Polygon
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_Polygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Polygon
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_Polygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Polygon
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_Polygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Polygon
      (alinestring ST_LineString)
      RETURNS ST_Polygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```



```
CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString,
    ansrid INTEGER)
   RETURNS ST_Polygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString,
    alinestringarray ST_LineString ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Polygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString,
    alinestringarray ST_LineString ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_Polygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_ExteriorRing()
   RETURNS ST_LineString,

OVERRIDING METHOD ST_ExteriorRing
   (acurve ST_Curve)
   RETURNS ST_Polygon,

OVERRIDING METHOD ST_InteriorRings()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_InteriorRings
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Polygon,

OVERRIDING METHOD ST_InteriorRingN
   (aposition INTEGER)
   RETURNS ST_LineString
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The *ST_Polygon* type provides for public use:

a) a method *ST_Polygon(CHARACTER LARGE OBJECT)*,
b) a method *ST_Polygon(CHARACTER LARGE OBJECT, INTEGER)*,
c) a method *ST_Polygon(BINARY LARGE OBJECT)*,
d) a method *ST_Polygon(BINARY LARGE OBJECT, INTEGER)*,
e) a method *ST_Polygon(ST_LineString)*,
f) a method *ST_Polygon(ST_LineString, INTEGER)*,
g) a method *ST_Polygon(ST_LineString, ST_LineString ARRAY)*,
h) a method *ST_Polygon(ST_LineString, ST_LineString ARRAY, INTEGER)*,
i) an overriding method *ST_ExteriorRing()*,
j) an overriding method *ST_ExteriorRing(ST_Curve)*,
k) an overriding method *ST_InteriorRings()*,
l) an overriding method *ST_InteriorRings(ST_Curve ARRAY)*,
m) an overriding method *ST_InteriorRingN(INTEGER)*,
n) a function *ST_PolyFromText(CHARACTER LARGE OBJECT)*,
o) a function *ST_PolyFromText(CHARACTER LARGE OBJECT, INTEGER)*,
p) a function *ST_PolyFromWKB(BINARY LARGE OBJECT)*,
q) a function *ST_PolyFromWKB(BINARY LARGE OBJECT, INTEGER)*,
r) a function *ST_PolyFromGML(CHARACTER LARGE OBJECT)*,
s) a function *ST_PolyFromGML(CHARACTER LARGE OBJECT, INTEGER)*,
t) a function *ST_BdPolyFromText(CHARACTER LARGE OBJECT)*,
u) a function *ST_BdPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*,
v) a function *ST_BdPolyFromWKB(BINARY LARGE OBJECT)*,
w) a function *ST_BdPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateExteriorRing* attribute is an *ST_LineString* value that is a linear ring.
3) The *ST_PrivateInteriorRings* attribute is a collection of *ST_LineString* values. Each *ST_LineString* value in the collection is a linear ring.
4) The linear ring in the *ST_PrivateExteriorRing* attribute and the linear rings in the *ST_PrivateInteriorRings* attribute represent the boundary of the *ST_Polygon* value.
5) An *ST_Polygon* value returned by the constructor function corresponds to the empty set.

8.3.2 ST_Polygon Methods

Purpose

Return an ST_Polygon value constructed from either the well-known text representation, the well-known binary representation, a GML representation, or the specified ST_LineString values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Polygon
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN NEW ST_Polygon(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_Polygon
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Polygon
   FOR ST_Polygon
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Polygon
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN NEW ST_Polygon(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_Polygon
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN ST_PolyFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString)
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN SELF.ST_SRID(0).ST_ExteriorRing(alinestring).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString,
    ansrid INTEGER)
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorRing(alinestring).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString,
    alinestringarray ST_LineString ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN SELF.ST_SRID(0).ST_ExteriorRing(alinestring).
      ST_InteriorRings(alinestringarray)
```

```
CREATE CONSTRUCTOR METHOD ST_Polygon
   (alinestring ST_LineString,
    alinestringarray ST_LineString ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_Polygon
   FOR ST_Polygon
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorRing(alinestring).
      ST_InteriorRings(alinestringarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_Polygon(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_Polygon(awktorgml, 0)*.

3) The method *ST_Polygon(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_Polygon(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Polygon or PolygonPatch XML element in the GML representation, then return the result of the value expression: *ST_PolyFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_PolyFromText(awktorgml, ansrid)*.

5) The method *ST_Polygon(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(BINARY LARGE OBJECT)* return the result of the value expression: *NEW ST_Polygon(awkb, 0)*.

7) The method *ST_Polygon(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_PolyFromWKB(awkb, ansrid)*.

9) The method *ST_Polygon(ST_LineString)* takes the following input parameters:

   a) an *ST_LineString* value *alinestring*.

10) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(ST_LineString)* returns an *ST_Polygon* value with:

   a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *alinestring*, the *ST_PrivateCoordinateDimension* attribute set to *alinestring.ST_PrivateCoordinateDimension* the *ST_PrivateDimension* attribute set to 2, the *ST_PrivateIs3D* attribute set to *alinestring.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured* attribute set to *alinestring.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_LineString* ARRAY value.

11) The method *ST_Polygon(ST_LineString, INTEGER)* takes the following input parameters:

a) an *ST_LineString* value *alinestring*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(ST_LineString, INTEGER)* returns an *ST_Polygon* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *alinestring*, the *ST_PrivateDimension* attribute set to *alinestring*.*ST_PrivateCoordinateDimension*, the ST_PrivateDimension attribute set to 2, the *ST_PrivateIs3D* attribute set to *alinestring.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured* attribute set to *alinestring.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_LineString* ARRAY value.

13) The method *ST_Polygon(ST_LineString, ST_LineString ARRAY)* takes the following input parameters:

a) an *ST_LineString* value *alinestring*,

b) an *ST_LineString* ARRAY value *alinestringarray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(ST_LineString, ST_LineString ARRAY)* returns an *ST_Polygon* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *alinestring*, the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(alinestring, alinestringarray)*, the *ST_PrivateDimension* attribute set to 2 the *ST_PrivateIs3D attribute* set to *alinestring.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured attribute* set to *alinestring.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to *alinestringarray*.

15) The method *ST_Polygon(ST_LineString, ST_LineString ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_LineString* value *alinestring*,

b) an *ST_LineString* ARRAY value *alinestringarray*,

c) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_Polygon(ST_LineString, ST_LineString ARRAY, INTEGER)* returns an *ST_Polygon* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *alinestring*, the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(alinestring, alinestringarray)*, the *ST_PrivateDimension* attribute set to 2**,** the *ST_PrivateIs3D attribute* set to *alinestring.ST_PrivateIs3D*, and the *ST_PrivateIsMeasured attribute* set to *alinestring.ST_PrivateIsMeasured*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to *alinestringarray*.

8.3.3 ST_ExteriorRing Methods

Purpose

Observe and mutate the ST_PrivateExteriorRing attribute of an ST_Polygon value.

Definition

```
CREATE METHOD ST_ExteriorRing()
   RETURNS ST_LineString
   FOR ST_Polygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateExteriorRing AS ST_LineString)
      END

CREATE METHOD ST_ExteriorRing
   (acurve ST_Curve)
   RETURNS ST_Polygon
   FOR ST_Polygon
   BEGIN
      -- If acurve is not an ST_LineString, then raise an exception
      IF acurve IS NOT OF (ST_LineString) THEN
         SIGNAL SQLSTATE '2FF12'
            SET MESSAGE_TEXT = 'curve value is not a linestring value';
      END IF;
      RETURN (SELF AS ST_CurvePolygon).ST_ExteriorRing(acurve);
   END
```

Description

1) The method *ST_ExteriorRing()* has no input parameters.

2) For the null-call method *ST_ExteriorRing()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateExteriorRing* attribute of SELF.

3) The method *ST_ExteriorRing(ST_Curve)* takes the following input parameters:

   a) an *ST_Curve* value *alinestring*.

4) For the type-preserving method *ST_ExteriorRing(ST_Curve)*:

   Case:

   a) If *acurve* is not an *ST_LineString* value, then an exception condition is raised: *SQL/MM Spatial exception – curve value is not a linestring value*.

   b) Otherwise, return an *ST_Polygon* value as a result of the value expression: *(SELF AS ST_CurvePolygon).ST_ExteriorRing(acurve).*

8.3.4 ST_InteriorRings Methods

Purpose

Observe and mutate the ST_PrivateInteriorRings attribute of an ST_Polygon value.

Definition

```
CREATE METHOD ST_InteriorRings()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_Polygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateInteriorRings AS
               ST_LineString ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_InteriorRings
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Polygon
   FOR ST_Polygon
   BEGIN
      DECLARE counter INTEGER;

      -- Check if curves are ST_LineString values
      SET counter = 1;
      WHILE counter <= CARDINALITY(acurvearray) DO
         -- If the current element is not an ST_LineString value, then
         -- raise an exception.
         IF acurvearray[counter] IS NOT OF (ST_LineString) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT = 'element is not an ST_LineString type';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      RETURN (SELF AS ST_CurvePolygon).ST_InteriorRings(acurvearray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_InteriorRings()* has no input parameters.

2) For the null-call method *ST_InteriorRings()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateInteriorRings* attribute of SELF.

3) The method *ST_InteriorRings(ST_Curve ARRAY)* takes the following input parameters:

   a) an *ST_Curve* ARRAY value *acurvearray*.

4) For the type-preserving method *ST_InteriorRings(ST_Curve ARRAY)*:

   Case:

   a) If any element in *acurvearray* is not an *ST_LineString* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_LineString type*.

b) Otherwise, return an *ST_Polygon* value as a result of the value expression: *(SELF AS ST_CurvePolygon).ST_InteriorRings(acurvearray).*

8.3.5 ST_InteriorRingN Method

Purpose

Return the specified element in the ST_PrivateInteriorRings attribute of an ST_Polygon value.

Definition

```
CREATE METHOD ST_InteriorRingN
   (aposition INTEGER)
   RETURNS ST_LineString
   FOR ST_Polygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            TREAT((SELF AS ST_CurvePolygon).ST_InteriorRingN(aposition) AS
               ST_LineString)
      END
```

Description

1) The method *ST_InteriorRingN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_InteriorRingN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return an *ST_LineString* value as a result of the value expression *TREAT((SELF AS ST_CurvePolygon).ST_InteriorRingN(aposition) AS ST_LineString)*.

8.3.6 ST_PolyFromText Functions

Purpose

Return an ST_Polygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Polygon value.

Definition

```
CREATE FUNCTION ST_PolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PolyFromText(awkt, 0)
```

```
CREATE FUNCTION ST_PolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_PolyFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awkt*.

2) The null-call function *ST_PolyFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_PolyFromText(awkt, 0)*.

3) The function *ST_PolyFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awkt*,

   b) an INTEGER value *ansrid*.

4) For the null-call function *ST_PolyFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Polygon* value.

      If *awkt* is not producible in the BNF for <polygon text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_Polygon)*.

8.3.7 ST_PolyFromWKB Functions

Purpose

Return an ST_Polygon value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Polygon value.

Definition

```
CREATE FUNCTION ST_PolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PolyFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_PolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_PolyFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*.

2) The null-call function *ST_PolyFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_PolyFromWKB(awkb, 0)*.

3) The function *ST_PolyFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

4) For the null-call function *ST_PolyFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_Polygon* value.

      If *awkb* is not producible in the BNF for <polygon binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_Polygon)*.

8.3.8 ST_PolyFromGML Functions

Purpose

Return an ST_Polygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Polygon or PolygonPatch representation of an ST_Polygon value.

Definition

```
CREATE FUNCTION ST_PolyFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PolyFromGML(agml, 0)

CREATE FUNCTION ST_PolyFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_PolyFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_PolyFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_PolyFromGML(agml, 0)*.
3) The function *ST_PolyFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_PolyFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a Polygon or PolygonPatch XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise,
      i) if any of the Polygon or PolygonPatch XML element Rings are not linear, convert them into their implementation-defined LinearRing approximations.
      ii) return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_Polygon)*.

8.3.9 ST_BdPolyFromText Functions

Purpose

Return an ST_Polygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_BdPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BdPolyFromText(awkt, 0)

CREATE FUNCTION ST_BdPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The function *ST_BdPolyFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_BdPolyFromText(CHARACTER LARGE OBJECT)* returns an *ST_Polygon* value as the result of the value expression: *ST_BdPolyFromText(awkt, 0)*.
3) The function *ST_BdPolyFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BdPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*:
   a) The parameter *awkt* is the well-known text representation of an *ST_MultiLineString* value. If *awkt* is not producible in the BNF for <multilinestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Use *ST_MLineFromText(CHARACTER LARGE OBJECT)* to transform *awkt* to an *ST_MultiLineString* value, *AMLS.*
   c) If any *ST_LineString* value in *AMLS* is not a linear ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   d) Using an implementation-dependent algorithm, an exterior linear ring, *ELR*, and an array of zero or more interior rings, *AILR*, are determined from the array of linear rings in *AMLS*.
   e) Return an *ST_Polygon* value with:
      i) The spatial reference system identifier set to *ansrid*.

ii) Using the method *ST_ExteriorRing(ST_LineString)*, the *ST_PrivateExteriorRing* attribute set to *ELR*.

iii) Using the method *ST_InteriorRings(ST_LineString ARRAY)*, the *ST_PrivateInteriorRings* attribute set to *AILR*.

8.3.10 ST_BdPolyFromWKB Functions

Purpose

Return an ST_MultiPolygon value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_BdPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BdPolyFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_BdPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The function *ST_BdPolyFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) For the null-call function *ST_BdPolyFromWKB(BINARY LARGE OBJECT)* returns an *ST_Polygon* value as the result of the value expression: *ST_BdPolyFromText(awkt, 0)*.
3) The function *ST_BdPolyFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BdPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*:
   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiLineString* value. If *awkb* is not producible in the BNF for <multilinestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Use *ST_MLineFromWKB(BINARY LARGE OBJECT)* to transform *awkb* to an *ST_MultiLineString* value, *AMLS*.
   c) If any *ST_LineString* value in *AMLS* is not a linear ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   d) Using an implementation-dependent algorithm, an exterior linear ring, *ELR*, and an array of zero or more interior rings, *AILR*, are determined from the array of linear rings in *AMLS*.
   e) Return an *ST_Polygon* value with:
      i) The spatial reference system identifier set to *ansrid*.

ii) Using the method *ST_ExteriorRing(ST_LineString)*, the *ST_PrivateExteriorRing* attribute set to *ELR*.

iii) Using the method *ST_InteriorRings(ST_LineString ARRAY)*, the *ST_PrivateInteriorRings* attribute set to *AILR*.

8.4 ST_Triangle Type and Routines

8.4.1 ST_Triangle Type

Purpose

The ST_Triangle type is a subtype of ST_Polygon with an exterior boundary having exactly four points and no interior boundaries.

Definition

```
CREATE TYPE ST_Triangle
   UNDER ST_Polygon
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Triangle
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_Triangle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Triangle
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_Triangle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Triangle
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_Triangle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Triangle
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_Triangle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Triangle
      (alinestring ST_LineString)
      RETURNS ST_Triangle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Triangle
   (alinestring ST_LineString,
    ansrid INTEGER)
   RETURNS ST_Triangle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Triangle
   (apointarray ST_Point ARRAY[4])
   RETURNS ST_Triangle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Triangle
   (apointarray ST_Point ARRAY[4],
    ansrid INTEGER)
   RETURNS ST_Triangle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points()
   RETURNS ST_Point ARRAY[4]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Points
   (apointarray ST_Point ARRAY[4])
   RETURNS ST_Triangle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_3DSlope()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_ExteriorRing()
   RETURNS ST_LineString,

OVERRIDING METHOD ST_ExteriorRing
   (acurve ST_Curve)
   RETURNS ST_Triangle,
```

```
OVERRIDING METHOD ST_InteriorRings()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_InteriorRings
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Triangle,

OVERRIDING METHOD ST_InteriorRingN
   (aposition INTEGER)
   RETURNS ST_LineString
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The *ST_Triangle* type provides for public use:
   a) a method *ST_Triangle(CHARACTER LARGE OBJECT)*,
   b) a method *ST_Triangle(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_Triangle(BINARY LARGE OBJECT)*,
   d) a method *ST_Triangle(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_Triangle(ST_LineString)*,
   f) a method *ST_Triangle(ST_LineString, INTEGER)*,
   g) a method *ST_Triangle(ST_Point ARRAY)*,
   h) a method *ST_Triangle(ST_Point ARRAY, INTEGER)*,
   i) a method *ST_Points()*,
   j) a method *ST_Points(ST_Point ARRAY)*,
   k) a method *ST_3DSlope()*,
   l) an overriding method *ST_ExteriorRing()*,
   m) an overriding method *ST_ExteriorRing(ST_Curve)*,
   n) an overriding method *ST_InteriorRings()*,
   o) an overriding method *ST_InteriorRings(ST_Curve ARRAY)*,
   p) an overriding method *ST_InteriorRingN(INTEGER)*,
   q) a function *ST_TriFromText(CHARACTER LARGE OBJECT)*,
   r) a function *ST_TriFromText(CHARACTER LARGE OBJECT, INTEGER)*,
   s) a function *ST_TriFromWKB(BINARY LARGE OBJECT)*,
   t) a function *ST_TriFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   u) a function *ST_TriFromGML(CHARACTER LARGE OBJECT)*,
   v) a function *ST_TriFromGML(CHARACTER LARGE OBJECT, INTEGER)*,
2) The *ST_PrivateExteriorRing* attribute is an *ST_LineString* value that is a linear ring with *ST_NumPoints* = 4.
3) The ST_Triangle value has no interior rings so the *ST_PrivateInteriorRings* attribute is set to an empty *ST_LineString* ARRAY value.

4) The linear ring in the *ST_PrivateExteriorRing* attribute represents the boundary of the *ST_Triangle* value.
5) An *ST_Triangle* value returned by the constructor function corresponds to the empty set.

8.4.2 ST_Triangle Methods

Purpose

Return an ST_Triangle value constructed from either the well-known text representation, the well-known binary representation, the GML representation, the specified ST_LineString value, or the specified ST_Point values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Triangle
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN NEW ST_Triangle(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_Triangle
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Triangle
   FOR ST_Triangle
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Triangle
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN NEW ST_Triangle(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_Triangle
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN ST_TriFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_Triangle
   (alinestring ST_LineString)
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN SELF.ST_SRID(0).ST_ExteriorRing(alinestring).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_Triangle
   (alinestring ST_LineString,
    ansrid INTEGER)
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorRing(alinestring).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_Triangle
   (apointarray ST_Point ARRAY)
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN SELF.ST_SRID(0).ST_ExteriorRing.ST_LineString(apointarray).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]))
```

```
CREATE CONSTRUCTOR METHOD ST_Triangle
   (alinestring ST_LineString,
    ansrid INTEGER)
   RETURNS ST_Triangle
   FOR ST_Triangle
   RETURN SELF.ST_SRID(ansrid).
      ST_ExteriorRing.ST_LineString(apointarray).
      ST_InteriorRings(CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]))
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_Triangle(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_Triangle(awktorgml, 0)*.
3) The method *ST_Triangle(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_Triangle(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Triangle XML element in the GML representation, then return the result of the value expression: *ST_TriFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_TriFromText(awktorgml, ansrid)*.
5) The method *ST_Triangle(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(BINARY LARGE OBJECT)* return the result of the value expression: *NEW ST_Triangle(awkb, 0)*.
7) The method *ST_Triangle(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
8) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_TriFromWKB(awkb, ansrid)*.
9) The method *ST_Triangle(ST_LineString)* takes the following input parameters:
   a) an *ST_LineString* value *alinestring*.
10) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(ST_LineString)* returns an *ST_Triangle* value with:
   a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *alinestring*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *alinestring.ST_CoordDim()*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_LineString* ARRAY value.

11) The method *ST_Triangle(ST_LineString, INTEGER)* takes the following input parameters:

a) an *ST_LineString* value *alinestring*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(ST_LineString, INTEGER)* returns an *ST_Triangle* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorRing(ST_Curve)*:

i) the *ST_PrivateDimension* attribute set to 2.

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression *ST_GetCoordDim(alinestring)*.

iii) the *ST_PrivateIs3D* attribute set to the value expression *alinestring.ST_Is3D()*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression *alinestring.ST_IsMeasured().*

v) the *ST_PrivateExteriorRing* attribute set to *alinestring.*

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_LineString* ARRAY value.

13) The method *ST_Triangle(ST_Point ARRAY)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(ST_Point ARRAY)* returns an *ST_Triangle* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorRing(ST_Curve)*:

i) the *ST_PrivateDimension* attribute set to 2.

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression *ST_GetCoordDim(apointarray)*.

iii) the *ST_PrivateIs3D* attribute set to the value expression *apointarray.ST_Is3D()*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression *ST_GetIsMeasured(apointarray).*

v) the *ST_PrivateExteriorRingf* attribute set to *ST_Linestring(apointarray)*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_LineString* ARRAY value.

15) The method *ST_Triangle(ST_Point ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Point* ARRAY value *apointarray*,

b) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_Triangle(ST_Point ARRAY, INTEGER)* returns an *ST_Triangle* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorRing(ST_Curve)*, the *ST_PrivateExteriorRing* attribute set to *ST_Linestring(apointarray)*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_Linestring(apointarray).ST_CoordDim()*.

c) Using the method *ST_InteriorRings(ST_Curve ARRAY)*, the *ST_PrivateInteriorRings* attribute set to an empty *ST_LineString* ARRAY value.

8.4.3 ST_Points Methods

Purpose

Observe and mutate the ST_PrivatePoints attribute of the ST_LineString value of the ST_PrivateExteriorRing attribute of an ST_Triangle value.

Definition

```
CREATE METHOD ST_Points()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_Triangle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateExteriorRing.ST_PrivatePoints
      END

CREATE METHOD ST_Points
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_LineString
   FOR ST_LineString
   BEGIN
      IF CARDINALITY(apointarray) <> 4 THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
       END IF;
       RETURN SELF.ST_ExteriorRing(NEW ST_LineString(apointarray));
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Points()* has no input parameters.

2) For the null-call method *ST_Points()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateExteriorRing* attribute of SELF.

3) The method *ST_Points(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*.

4) For the type-preserving method *ST_Points(ST_Point ARRAY)*:

   a) If *apointarray* does not have exactly four elements, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   b) Otherwise, return an *ST_Triangle* value with the exterior ring set to *ST_LineString*(*apointarray*,SELF.*ST_SRID()*).

8.4.4 ST_3DSlope Method

Purpose

Return the slope of an ST_Triangle value as a ratio.

Definition

```
CREATE METHOD ST_3DSlope()
   RETURNS DOUBLE PRECISION
   FOR ST_Triangle
   BEGIN
   --
   -- See Description
   --
   END
```

Description

1) The method *ST_3DSlope()* has no input parameters.

2) For the null-call method *ST_3DSlope()*:

   a) Case:

      i) If SELF is an empty set, then return the null value.

      ii) If *SELF.ST_Is3D()* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial Exception – geometry needs to be 3D*.

      iii) Otherwise, return the implementation-defined slope of SELF, such that z coordinate values are considered in the calculation.

8.4.5 ST_ExteriorRing Methods

Purpose

Observe and mutate the ST_PrivateExteriorRing attribute of an ST_Triangle value.

Definition

```
CREATE METHOD ST_ExteriorRing()
   RETURNS ST_LineString
   FOR ST_Triangle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateExteriorRing AS ST_LineString)
      END

CREATE METHOD ST_ExteriorRing
   (acurve ST_Curve)
   RETURNS ST_Triangle
   FOR ST_Triangle
   BEGIN
      -- If acurve is not an ST_LineString, then raise an exception
      IF acurve IS NOT OF (ST_LineString) THEN
         SIGNAL SQLSTATE '2FF12'
            SET MESSAGE_TEXT = 'curve value is not a linestring value';
      END IF;
      -- If acurve is not an ST_LineString having exactly 4 points,
      -- then raise an exception
      IF (TREAT(acurve AS ST_LineString).ST_NumPoints() <> 4) THEN SIGNAL
      SQLSTATE '2FF75'
         SET MESSAGE_TEXT = 'exterior ring must have exactly 4 points';
      END IF;
      RETURN (SELF AS ST_CurvePolygon).ST_ExteriorRing(acurve);
   END
```

Description

1) The method *ST_ExteriorRing()* has no input parameters.
2) For the null-call method *ST_ExteriorRing()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateExteriorRing* attribute of SELF.
3) The method *ST_ExteriorRing(ST_Curve)* takes the following input parameters:
   a) an *ST_Curve* value alinestring.
4) For the type-preserving method *ST_ExteriorRing(ST_Curve)*:

   Case:

   a) If acurve is not an *ST_LineString* value, then an exception condition is raised: *SQL/MM Spatial exception – curve value is not a linestring value*.
   b) If acurve is not an *ST_LineString* value having exactly four points, then an exception condition is raised: *SQL/MM Spatial exception – exterior ring must have exactly 4 points*.
   c) Otherwise, return an *ST_Triangle* value as a result of the value expression: *(SELF AS ST_CurvePolygon).ST_ExteriorRing(acurve).*

8.4.6 ST_InteriorRings Methods

Purpose

Observe and mutate the ST_PrivateInteriorRings attribute of an ST_Triangle value.

Definition

```
CREATE METHOD ST_InteriorRings()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_Triangle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateInteriorRings AS
               ST_LineString ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_InteriorRings
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Triangle
   FOR ST_Triangle
   BEGIN
      -- Check if acurvearray is empty
      IF CARDINALITY(acurvearray) > 0 THEN
         SIGNAL SQLSTATE '2FF66'
            SET MESSAGE_TEXT = 'triangles cannot have holes';
      END IF;
      RETURN (SELF AS ST_CurvePolygon).ST_InteriorRings(acurvearray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_InteriorRings()* has no input parameters.
2) For the null-call method *ST_InteriorRings()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateInteriorRings* attribute of SELF.
3) The method *ST_InteriorRings(ST_Curve ARRAY)* takes the following input parameters:

   a) an *ST_Curve* ARRAY value *acurvearray*.
4) For the type-preserving method *ST_InteriorRings(ST_Curve ARRAY)*:

   Case:

   a) If *acurvearray* contains any elements, then an exception condition is raised: *SQL/MM Spatial exception – triangles cannot have holes*.
   b) Otherwise, return an *ST_Triangle* value as a result of the value expression: *(SELF AS ST_CurvePolygon).ST_InteriorRings(acurvearray).*

8.4.7 ST_InteriorRingN Method

Purpose

Return the specified element in the ST_PrivateInteriorRings attribute of an ST_Triangle value.

Definition

```
CREATE METHOD ST_InteriorRingN
   (aposition INTEGER)
   RETURNS ST_LineString
   FOR ST_Triangle
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SIGNAL SQLSTATE '2FF66'
               SET MESSAGE_TEXT = 'triangles cannot have holes'
      END
```

Description

1) The method *ST_InteriorRingN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_InteriorRingN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, an exception condition is raised: *SQL/MM Spatial exception – triangles cannot have holes*.

8.4.8 ST_TriFromText Functions

Purpose

Return an ST_Triangle value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Triangle value.

Definition

```
CREATE FUNCTION ST_TriFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Triangle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_TriFromText(awkt, 0)

CREATE FUNCTION ST_TriFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_Triangle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_TriFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_TriFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_TriFromText(awkt, 0)*.
3) The function *ST_TriFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_TriFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Triangle* value.

      If *awkt* is not producible in the BNF for <triangle text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_Triangle)*.

8.4.9 ST_TriFromWKB Functions

Purpose

Return an ST_Triangle value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Triangle value.

Definition

```
CREATE FUNCTION ST_TriFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Triangle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_TriFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_TriFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_Triangle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_TriFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_TriFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_TriFromWKB(awkb, 0)*.
3) The function *ST_TriFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_TriFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_Triangle* value.

      If *awkb* is not producible in the BNF for <triangle binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_Triangle)*.

8.4.10 ST_TriFromGML Functions

Purpose

Return an ST_Triangle value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Triangle value.

Definition

```
CREATE FUNCTION ST_TriFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_Triangle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_TriFromGML(agml, 0)

CREATE FUNCTION ST_TriFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_Triangle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_TriFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_TriFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_TriFromGML(agml, 0)*.
3) The function *ST_TriFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_TriFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a Triangle XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_Triangle)*.

8.5 ST_PolyhdrlSurface Type and Routines

8.5.1 ST_PolyhdrlSurface Type

Purpose

The ST_PolyhdrlSurface type is a subtype of ST_Surface composed of contiguous polygon surfaces (ST_Polygon) connected along their common boundary curves. This differs from ST_Surface only in the restriction on the types of surface patches acceptable.

Definition

```
CREATE TYPE ST_PolyhdrlSurface
   UNDER ST_Surface
   AS (
      ST_PrivatePatches ST_Polygon
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_PolyhdrlSurface
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_PolyhdrlSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_PolyhdrlSurface
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_PolyhdrlSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_PolyhdrlSurface
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_PolyhdrlSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_PolyhdrlSurface
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_PolyhdrlSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_PolyhdrlSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Patches()
   RETURNS ST_Polygon ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Patches
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_PolyhdrlSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumPatches()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PatchN
   (aposition INTEGER)
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivatePatches* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePatches*.

Description

1) The *ST_PolyhdrlSurface* type provides for public use:
   a) a method *ST_PolyhdrlSurface(CHARACTER LARGE OBJECT)*,
   b) a method *ST_PolyhdrlSurface(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_PolyhdrlSurface(BINARY LARGE OBJECT)*,
   d) a method *ST_PolyhdrlSurface(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_PolyhdrlSurface(ST_Polygon ARRAY)*,
   f) a method *ST_PolyhdrlSurface(ST_Polygon ARRAY, INTEGER)*,
   g) a method *ST_Patches()*,
   h) a method *ST_Patches(ST_Polygon ARRAY)*,
   i) a method *ST_NumPatches()*,
   j) a method *ST_PatchN(INTEGER)*,
   k) a function *ST_PhSFromText(CHARACTER LARGE OBJECT)*,
   l) a function *ST_PhSFromText(CHARACTER LARGE OBJECT, INTEGER)*,
   m) a function *ST_PhSFromWKB(BINARY LARGE OBJECT)*,
   n) a function *ST_PhSFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   o) a function *ST_PhSFromGML(CHARACTER LARGE OBJECT)*,
   p) a function *ST_PhSFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The *ST_PrivatePatches* attribute is a collection of contiguous *ST_Polygon* values connected along their common boundary curves.
3) The *ST_PrivatePatches* attribute shall not be the null value. The elements in the *ST_PrivatePatches* attribute shall not be the null value.
4) An *ST_PolyhdrlSurface* is contiguous.
5) An *ST_PolyhdrlSurface* is a topologically closed point set.
6) An *ST_PolyhdrlSurface* is a simple, closed polyhedron and is topologically isomorphic to the surface of a sphere.
7) If an *ST_PolyhdrlSurface* value is closed, then it bounds a solid.
8) All of the *ST_Polygon* values in the *ST_PrivatePatches* attribute shall be in the same spatial reference system as the *ST_PolyhdrlSurface* value.
9) The coordinate dimension of an *ST_PolyhdrlSurface* value is the number of coordinate values associated with the *ST_Polygon* values in the *ST_PrivatePatches* attribute having the lowest coordinate dimension.
10) The rings in the boundary may spatially intersect at most only a single point:

$$\forall\, p \in \mathit{ST\_PolyhdrlSurface},\ \forall\, c_1, c_2 \in \mathrm{Boundary}(p),\ c_1 \neq c_2,$$
$$\forall\, a_1, a_2 \in \mathit{ST\_Point},\ a_1, a_2 \in c_1,\ a_1 \neq a_2,\ [\, a_1 \in c_2 \Rightarrow a_2 \notin c_2 \,]$$

11) An *ST_PolyhdrlSurface* value shall not have cut lines, spikes or punctures:

$$\forall\, p \in \mathit{ST\_PolyhdrlSurface},\ p = \mathrm{Closure}(\mathrm{Interior}(p))$$

12) The interior of every *ST_PolyhdrlSurface* value is a connected point set.
13) The exterior of an *ST_PolyhdrlSurface* with one or more holes is not connected. Each hole defines a disconnected component of the exterior.
14) An *ST_PolyhdrlSurface* value returned by the constructor function corresponds to the empty set.
15) An *ST_PolyhdrlSurface* value corresponds to the empty set if the *ST_PrivatePatches* attribute corresponds to the empty set.

16) An *ST_PolyhdrlSurface* value is well formed only if all of the *ST_Polygon* values in the *ST_PrivatePatches* attribute are well formed.

17) For all *ST_Polygon* values in the *ST_PrivatePatches* attribute, *ST_Polygon.Is3D* = 1 (one).

18) For each pair of *ST_Polygon* values in the *ST_PrivatePatches* attribute that "touch", the common boundary shall be expressible as a finite collection of *ST_LineStrings*. Each such *ST_LineString* shall be part of the boundary of at most 2 *ST_Polygon* values in the *ST_PrivatePatches* attribute.

19) For any two *ST_Polygon* values in the *ST_PrivatePatches* attribute that share a common boundary, the "top" of the *ST_Polygon* values shall be consistent. This means that when two linear rings from these two *ST_Polygon* values traverse the common boundary segment, they do so in opposite directions.

20) If all of the *ST_Polygon* values in the *ST_PrivatePatches* attribute are in alignment (that is, if their normals are parallel), then the whole stitched *ST_PolyhdrlSurface* value is co-planar and can be represented as a single patch (*ST_Polygon* value).

8.5.2 ST_PolyhdrlSurface Methods

Purpose

Return an ST_PolyhdrlSurface value constructed from either the well-known text representation, the well-known binary representation, a GML representation, or the specified ST_Polygon values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
   RETURN NEW ST_PolyhdrlSurface(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
   RETURN NEW ST_PolyhdrlSurface(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
   RETURN ST_PhSFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
   RETURN SELF.ST_SRID(0).ST_Patches(apolygonarray)

CREATE CONSTRUCTOR METHOD ST_PolyhdrlSurface
   (apolygonarray ST_Polygon  ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
      RETURN SELF.ST_SRID(ansrid). ST_Patches(apolygonarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_PolyhdrlSurface(CHARACTER LARGE OBJECT)* takes the following input parameter:

a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_PolyhdrlSurface(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_PolyhdrlSurface(awktorgml, 0)*.

3) The method *ST_PolyhdrlSurface(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a CHARACTER LARGE OBJECT value *awktorgml*,

b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_PolyhdrlSurface(CHARACTER LARGE OBJECT, INTEGER)*:

Case:

a) If *awktorgml* contains a PolyhedralSurface or PolygonPatch XML element in the GML representation, then return the result of the value expression: *ST_PhSFromGML(awktorgml, ansrid)*.

b) Otherwise, return the result of the value expression: *ST_PhSFromText(awktorgml, ansrid)*.

5) The method *ST_PolyhdrlSurface(BINARY LARGE OBJECT)* takes the following input parameter:

a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_PolyhdrlSurface(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_PolyhdrlSurface(awkb, 0)*.

7) The method *ST_PolyhdrlSurface(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a BINARY LARGE OBJECT value *awkb*,

b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_PolyhdrlSurface(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_PhSFromWKB(awktorgml, ansrid)*.

9) The method *ST_PolyhdrlSurface(ST_Polygon ARRAY)* takes the following input parameters:

a) an *ST_Polygon* ARRAY value *apolygonarray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_PolyhdrlSurface(ST_Polygon ARRAY)* returns an *ST_PolyhdrlSurface* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_Patches(ST_Polygon ARRAY)*:

i) the *ST_PrivateDimension* attribute set to 2.

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression *ST_GetCoordDim(apolygonarray)*.

iii) the *ST_PrivateIs3D* attribute set to 1 (one).

iv) the *ST_PrivateIsMeasured* attribute set to the value expression *ST_GetIsMeasured(apolygonarray).*

v) the *ST_PrivatePatches* attribute set to *apolygonarray*.

11) The method *ST_PolyhdrlSurface(ST_Polygon ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Polygon* ARRAY value *apolygonarray*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_PolyhdrlSurface(ST_Polygon ARRAY, INTEGER)* returns an *ST_PolyhdrlSurface* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Patches(ST_Polygon ARRAY)*:

   i) the *ST_PrivateDimension* attribute set to 2.

   ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression *ST_GetCoordDim(apolygonarray)*.

   iii) the *ST_PrivateIs3D* attribute set to 1 (one).

   iv) the *ST_PrivateIsMeasured* attribute set to the value expression *ST_GetIsMeasured(apolygonarray).*

   v) the *ST_PrivatePatches* attribute set to *apolygonarray*.

8.5.3 ST_Patches Methods

Purpose

Observe and mutate the ST_PrivatePatches attribute of an ST_PolyhdrlSurface value.

Definition

```
CREATE METHOD ST_Patches()
   RETURNS ST_Polygon ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_PolyhdrlSurface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivatePatches
      END

CREATE METHOD ST_Patches
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_PolyhdrlSurface
   FOR ST_PolyhdrlSurface
   BEGIN
      DECLARE acounter INTEGER;
      DECLARE anothercounter INTEGER;
      DECLARE yetanothercounter INTEGER;
      DECLARE test INTEGER;
      DECLARE intersection ST_Geometry;

      -- If apolygonarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(apolygonarray);
      -- If apolygonarray is not 3D, then raise an exception.
      IF ST_GetIs3D(apolygonarray) = 0 THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument'
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN SELF;
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and apolygonarray.
      IF (CARDINALITY(apolygonarray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(apolygonarray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- For all patches, check that they are contiguous with another
      -- patch.  If so, check that the intersecting boundary does not
      -- overlap the boundary of another patch
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(apolygonarray) DO
         -- Every polygon must intersect the boundary of at least one
         -- other polygon in the input array such that the intersection
         -- is a (multi)linestring.  If not, raise an exception
         SET test = 0;
         SET anothercounter = 1;
         WHILE anothercounter <= CARDINALITY(apolygonarray) AND test = 0
            DO
            IF anothercounter <> acounter THEN
```

```
            IF apolygonarray[acounter].ST_3DIntersects
               (apolygonarray[anothercounter]) = 1 THEN
               IF apolygonarray[acounter].ST_3DIntersection
                  (apolygonarray[anothercounter]) IS OF (ST_Curve)
                  OR apolygonarray[acounter].ST_3DIntersection
                  (apolygonarray[anothercounter]) IS OF (ST_MultiCurve)
                  THEN
                     SET test = 1;
               END IF;
            END IF;
         END IF;
         SET anothercounter = anothercounter + 1;
      END WHILE;
      IF test = 0 THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET acounter = acounter + 1;
   END WHILE;
   SET acounter = 1;
   WHILE acounter <= CARDINALITY(apolygonarray) DO
      -- The interior of the linestring of intersection between any two
      -- polygons in the input array must not intersect with the
      -- boundary of any other polygon in the input array. If not,
      -- raise an exception
      SET test = 0;
      SET anothercounter = 1;
      WHILE anothercounter <= CARDINALITY(apolygonarray) AND test = 0
         DO
         IF anothercounter <> acounter THEN
            IF apolygonarray[acounter].ST_3DIntersects
               (apolygonarray[anothercounter]) = 1 THEN
               SET intersection =
                  apolygonarray[acounter].ST_3DIntersection
                     (apolygonarray[anothercounter]);
               SET yetanothercounter = 1;
               WHILE yetanothercounter <= CARDINALITY(apolygonarray) DO
                  IF yetanothercounter <> acounter AND
                     yetanothercounter <> anothercounter THEN
                     IF intersection.ST_3DIntersects
                        (apolygonarray[yetanothercounter]) = 1 AND
                        intersection.ST_3DIntersection
                        (apolygonarray[yetanothercounter]).ST_Dimension
                        > 0 THEN
                        SET test = 1;
                     END IF;
                  END IF;
               END WHILE;
            END IF;
         END IF;
         SET anothercounter = anothercounter + 1;
      END WHILE;
      IF test = 1;
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET acounter = acounter + 1;
   END WHILE;
   -- Return an ST_PolyhdrlSurface value with the ST_PrivatePatches
   -- attribute set to apolygonarray.
   RETURN
```

```
        SELF.ST_PrivateDimension(2).
           ST_PrivateCoordinateDimension(ST_GetCoordDim(apolygonarray)).
           ST_PrivateIs3D(1).
           ST_PrivateIsMeasured(ST_GetIsMeasured(apolygonarray)).
           ST_PrivatePatches(apolygonarray);
  END
```

Description

1) The method *ST_Patches()* has no input parameters.
2) For the null-call method *ST_Patches()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivatePatches* attribute of SELF.
3) The method *ST_Patches(ST_Polygon ARRAY)* takes the following input parameters:
   a) an *ST_Polygon* ARRAY value *apolygonarray*.
4) For the type-preserving method *ST_Patches(ST_Polygon ARRAY)*:

   Case:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *apolygonarray* is the null value or contains null elements.
   b) If *apolygonarray* is not 3D, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   c) If SELF is the null value, then return the null value.
   d) If the cardinality of *apolygonarray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID*(*apolygonarray*), then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.
   e) Let P1 and P2 be two different *ST_Polygon* values in *apolygonarray.* If, for every P1 in apolygonarray, there does not exist a P2 such that the intersection of their boundaries is a line, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   f) Let P3 be an *ST_Polygon* value in *apolygonarray* different from P1 and P2*.* Let L1 be the (multi)line intersection of the boundaries of P1 and P2. If L1 intersects the boundary of any P3 with dimension of the result greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   g) Otherwise, return an *ST_PolyhdrlSurface* value with:
      i) The dimension set to 2.
      ii) The coordinate dimension set to the value expression: *ST_GetCoordDim(apolygonarray)*.
      iii) The *ST_PrivateIs3D* attribute set to 1 (one).
      iv) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apolygonarray)*.
      v) The *ST_PrivatePatches* attribute set to *apolygonarray*.

8.5.4 ST_NumPatches Method

Purpose

Return the cardinality of the ST_PrivatePatches attribute of an ST_PolyhdrlSurface value.

Definition

```
CREATE METHOD ST_NumPatches()
   RETURNS INTEGER
   FOR ST_PolyhdrlSurface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivatePatches)
      END
```

Description

1) The method *ST_NumPatches()* has no input parameters.
2) For the null-call method *ST_NumPatches()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the cardinality of the *ST_PrivatePatches* attribute.

8.5.5 ST_PatchN Method

Purpose

Return the specified element in the ST_PrivatePatches attribute of an ST_PolyhdrlSurface value.

Definition

```
CREATE METHOD ST_PatchN
   (aposition INTEGER)
   RETURNS ST_Polygon
   FOR ST_PolyhdrlSurface
   BEGIN
      IF SELF.ST_IsEmpty() = 1  THEN
         RETURN CAST (NULL AS ST_Polygon);
      END IF;
      IF aposition < 1 OR
         aposition > CARDINALITY(SELF.ST_PrivatePatches) THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Polygon);
         END;
      END IF;
      RETURN SELF.ST_PrivatePatches[aposition];
   END
```

Description

1) The method *ST_PatchN(INTEGER)* takes the following input parameters:

   a) an INTEGER value *aposition*.

2) For the null-call method *ST_PatchN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If *aposition* is less than one or greater than the cardinality of the *ST_PrivatePatches* attribute, then:

      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.

      ii) Return the null value.

   c) Otherwise, return an *ST_Polygon* value at element *aposition* in the *ST_PrivatePatches* attribute of SELF.

8.5.6 ST_PhSFromText Functions

Purpose

Return an ST_PolyhdrlSurface value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_PolyhdrlSurface value.

Definition

```
CREATE FUNCTION ST_PhSFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PhSFromText(awkt, 0)

CREATE FUNCTION ST_PhSFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_PhSFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awkt*.

2) The null-call function *ST_PhSFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_PhSFromText(awkt, 0)*.

3) The function *ST_PhSFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awkt*,

   b) an INTEGER value *ansrid*.

4) For the null-call function *ST_PhSFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_PolyhdrlSurface* value.

      If *awkt* is not producible in the BNF for <polyhedralsurface text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_PolyhdrlSurface)*.

8.5.7 ST_PhSFromWKB Functions

Purpose

Return an ST_PolyhdrlSurface value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_PolyhdrlSurface value.

Definition

```
CREATE FUNCTION ST_PhSFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PhSFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_PhSFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_PhSFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_PhSFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_PhSFromWKB(awkb, 0)*.
3) The function *ST_PhSFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_PhSFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_PolyhdrlSurface* value.

      If *awkb* is not producible in the BNF for <polyhedralsurface binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_PolyhdrlSurface)*.

8.5.8 ST_PhSFromGML Functions

Purpose

Return an ST_PolyhdrlSurface value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML PolyhedralSurface or PolygonPatch representation of an ST_PolyhdrlSurface value.

Definition

```
CREATE FUNCTION ST_PhSFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_PhSFromGML(agml, 0)

CREATE FUNCTION ST_PhSFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_PolyhdrlSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_PhSFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_PhSFromGML*(CHARACTER LARGE OBJECT) returns the result of the value expression: *ST_PhSFromGML(agml, 0)*.
3) The function *ST_PhSFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_PhSFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a PolyhedralSurface or PolygonPatch XML element in the GML representation , then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_PolyhdrlSurface)*.

8.6 ST_TIN Type and Routines

8.6.1 ST_TIN Type

Purpose

The ST_TIN type is a subtype of ST_Surface composed of contiguous triangle surfaces (ST_Triangle) connected along their common boundary linestrings. This differs from ST_PolyhdrlSurface in the restriction on the types of surface patches acceptable and the addition of TIN elements which constrain the triangulation.

Definition

```
CREATE TYPE ST_TIN
   UNDER ST_PolyhdrlSurface
   AS (
      ST_PrivateElements ST_TINElement
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[],
      ST_PrivateMaxSideLength DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_TIN
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_TIN
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_TIN
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_TIN
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_TIN
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_TIN
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_TIN
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_TIN
   (triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements],
    elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_TIN
   (triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements],
    elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_TIN
   (elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_TIN
   (elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_TINElements()
   RETURNS ST_TINElement ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_TINElements
   (elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    triangulate INTEGER)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_MaxSideLength()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MaxSideLength
   (maxsidelength DOUBLE PRECISION,
    triangulate INTEGER)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_TINTable()
   RETURNS TABLE
   (item CHARACTER VARYING(30),
    ordernumber INTEGER,
    xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    triangle INTEGER ARRAY[3],
    visibility INTEGER,
    elementID INTEGER,
    elementtag CHARACTER VARYING(64),
    element INTEGER ARRAY[ST_MaxIntegerArrayElements])
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_TINTable
   (tin_table_name DT1,
    item_column DT2,
    ordernumber_column DT2,
    xcoord_column DT2,
    ycoord_column DT2,
    zcoord_column DT2,
    triangle_column DT2,
    visibility_column DT2,
    elementID_column DT2,
    elementtag_column DT2,
    element_column DT2,
    maxsidelength DOUBLE PRECISION)
   RETURNS ST_TIN
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
```

```
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_Clip
      (clipboundary ST_Polygon)
      RETURNS ST_TIN
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   OVERRIDING METHOD ST_Patches()
      RETURNS ST_Triangle ARRAY[ST_MaxGeometryArrayElements],

   OVERRIDING METHOD ST_Patches
      (triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements])
      RETURNS ST_TIN
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) *ST_MaxIntegerArrayElements* is the implementation-defined maximum cardinality of an array of *INTEGER* elements.
5) *DT1* is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.
6) *DT2* is data type of variable-length character string with character set SQL_IDENTIFIER and maximum length not less than 128 characters.
7) The attribute *ST_PrivatePatches* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePatches*.
8) The attribute *ST_PrivateElements* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateElements*.
9) The attribute *ST_PrivateMaxSideLength* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateMaxSideLength*.

Description

1) The *ST_TIN* type provides for public use:
   a) a method *ST_TIN(CHARACTER LARGE OBJECT)*,
   b) a method *ST_TIN(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_TIN(BINARY LARGE OBJECT)*,
   d) a method *ST_TIN(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_TIN(ST_Triangle ARRAY, ST_TINElement ARRAY, DOUBLE PRECISION)*,
   f) a method *ST_TIN(ST_Triangle ARRAY, ST_TINElement ARRAY, DOUBLE PRECISION, INTEGER)*,
   g) a method *ST_TIN(ST_TINElement ARRAY, DOUBLE PRECISION)*,
   h) a method *ST_TIN(ST_TINElement ARRAY, DOUBLE PRECISION, INTEGER)*,
   i) a method *ST_TINElements()*,

j) a method *ST_TINElements(ST_TINElement ARRAY, INTEGER)*,

k) a method *ST_MaxSideLength()*,

l) a method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*,

m) a method *ST_TINTable()*,

n) a method *ST_TINTable(DT1, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DOUBLE PRECISION)*,

o) a method *ST_Clip(ST_Polygon)*,

p) an overriding method *ST_Patches()*,

q) an overriding method *ST_Patches(ST_Triangle ARRAY)*,

r) a function *ST_TINFromText(CHARACTER LARGE OBJECT)*,

s) a function *ST_TINFromText(CHARACTER LARGE OBJECT, INTEGER)*,

t) a function *ST_TINFromWKB(BINARY LARGE OBJECT)*,

u) a function *ST_TINFromWKB(BINARY LARGE OBJECT, INTEGER)*,

v) a function *ST_TINFromGML(CHARACTER LARGE OBJECT)*,

w) a function *ST_TINFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivatePatches* attribute is a collection of contiguous *ST_Triangle* values connected along their common boundary curves.

3) An *ST_TIN* value returned by the constructor function corresponds to the empty set.

4) The *ST_PrivateElements* attribute is a collection of *ST_TINElement* values representing constraints from which the TIN surface can be generated.

5) The *ST_PrivateMaxSideLength* attribute restricts the maximum length of each side of an *ST_Triangle* value that is an element in the *ST_PrivatePatches* attribute.

6) It is implementation-defined whether the restriction imposed by the *ST_PrivateMaxSideLength* attribute applies to all of the *ST_Triangle* values in the *ST_PrivatePatches* attribute or just those which lie along the boundary of the TIN surface.

7) The spatial reference systems of the *ST_Triangle* values in the *ST_PrivatePatches* attribute and of all the *ST_Geometry* values contained in the *ST_TINElements*values in the *ST_PrivateElements* attribute shall be equal to the spatial reference system of their *ST_TIN* value.

8.6.2 ST_TIN Methods

Purpose

Return an ST_TIN value constructed from either the well-known text representation; the well-known binary representation; the GML representation; the specified ST_Triangle triangle values, ST_TINElement TIN element values and the DOUBLE PRECISION maximum side length value; or the specified ST_TINElement TIN element values and the DOUBLE PRECISION maximum side length value.

Definition

```
CREATE CONSTRUCTOR METHOD ST_TIN
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_TIN
   FOR ST_TIN
   RETURN NEW ST_TIN(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_TIN
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_TIN
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_TIN
   FOR ST_TIN
   RETURN NEW ST_TIN(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_TIN
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_TIN
   FOR ST_TIN
   RETURN ST_TINFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_TIN
   (triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements],
    elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION)
   RETURNS ST_TIN
   FOR ST_TIN
   RETURN SELF.ST_SRID(0).
      ST_Patches(triangles).
      ST_TINElements(elements,0).
      ST_MaxSideLength(maxsidelength,0)

CREATE CONSTRUCTOR METHOD ST_TIN
   (triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements],
    elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_TIN
   FOR ST_TIN
   RETURN SELF.ST_SRID(ansrid).
      ST_Patches(triangles).
      ST_TINElements(elements,0).
      ST_MaxSideLength(maxsidelength,0)
```

```
CREATE CONSTRUCTOR METHOD ST_TIN
   (elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      --
      -- Create ST_Triangle ARRAY triangles from TIN elements
      -- and maxsidelength
      --
      -- See Description
      --
   RETURN SELF.ST_SRID(0).
      ST_Patches(triangles).
      ST_TINElements(elements,0).
      ST_MaxSideLength(maxsidelength,0)
   END

CREATE CONSTRUCTOR METHOD ST_TIN
   (elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    maxsidelength DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      --
      -- Create ST_Triangle ARRAY triangles from TIN elements
      -- and maxsidelength
      --
      -- See Description
      --
   RETURN SELF.ST_SRID(ansrid).
      ST_Patches(triangles).
      ST_TINElements(elements,0).
      ST_MaxSideLength(maxsidelength,0)
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_TIN(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_TIN(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_TIN(awktorgml, 0)*.
3) The method *ST_TIN(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_TIN(CHARACTER LARGE OBJECT, INTEGER)*:

Case:

a) If *awktorgml* contains a TIN XML element in the GML representation, then return the result of the value expression: *ST_TINFromGML(awktorgml, ansrid)*.

b) Otherwise, return the result of the value expression: *ST_TINFromText(awktorgml, ansrid)*.

5) The method *ST_TIN(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_TIN(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_TIN(awkb, 0)*.

7) The method *ST_TIN(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_TIN(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_TINFromWKB(awktorgml, ansrid)*.

9) The method *ST_TIN(ST_Triangle ARRAY, ST_TINElement ARRAY, DOUBLE PRECISION)* takes the following input parameters:

   a) an *ST_Triangle* ARRAY value *triangles*,

   b) an *ST_TINElement* ARRAY value *elements*,

   c) a DOUBLE PRECISION value *maxsidelength*.

10) The null-call type-preserving SQL-invoked constructor method *ST_TIN(ST_Triangle ARRAY, ST_TINElement ARRAY, DOUBLE PRECISION)* returns an *ST_TIN* value with:

   a) The spatial reference system identifier set to 0 (zero).

   b) Using the method *ST_Patches(ST_Triangle ARRAY)*, the *ST_PrivatePatches* attribute set to *triangles*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(triangles)*.

   c) Using the method *ST_TINElements(ST_TINElement ARRAY, INTEGER)*, the *ST_PrivateElements* attribute set to *elements*.

   d) Using the method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*, the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

11) The method *ST_TIN(ST_Triangle ARRAY, ST_TINElement ARRAY, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) an *ST_Triangle* ARRAY value *triangles*,

   b) an *ST_TINElement* ARRAY value *elements*,

   c) a DOUBLE PRECISION value *maxsidelength*,

   d) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_TIN(ST_Triangle ARRAY, ST_TINElement ARRAY, DOUBLE PRECISION, INTEGER)* returns an *ST_TIN* value with:

   a) The spatial reference system identifier set to *ansrid*.

   b) Using the method *ST_Patches(ST_Triangle ARRAY)*, the *ST_PrivatePatches* attribute set to *triangles*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(triangles)*.

   c) Using the method *ST_TINElements(ST_TINElement ARRAY, INTEGER)*, the *ST_PrivateElements* attribute set to *elements*.

   d) Using the method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*, the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

13) The method *ST_TIN(ST_TINElement ARRAY, DOUBLE PRECISION)* takes the following input parameters:

   a) an *ST_TINElement* ARRAY value *elements*,

   b) a DOUBLE PRECISION value *maxsidelength*.

14) The null-call type-preserving SQL-invoked constructor method *ST_TIN(ST_TINElement ARRAY, DOUBLE PRECISION)* returns an *ST_TIN* value with:

   a) The spatial reference system identifier set to 0 (zero).

   b) Using the method *ST_Patches(ST_Triangle ARRAY)*, the *ST_PrivatePatches* attribute set to *triangles*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(triangles)* where *triangles* is an *ST_Triangle* ARRAY obtained by applying the implementation-defined triangulation algorithm to the *ST_TINElement* ARRAY *elements* and constrained or modified by *maxsidelength*.

   c) Using the method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*, the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

15) The method *ST_TIN(ST_TINElement ARRAY, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) an *ST_TINElement* ARRAY value *elements*,

   b) a DOUBLE PRECISION value *maxsidelength*,

   c) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_TIN(ST_TINElement ARRAY, DOUBLE PRECISION, INTEGER)* returns an *ST_TIN* value with:

   a) The spatial reference system identifier set to *ansrid*.

   b) Using the method *ST_Patches(ST_Triangle ARRAY)*, the *ST_PrivatePatches* attribute set to *triangles*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(triangles)* where *triangles* is an *ST_Triangle* ARRAY obtained by applying the implementation-defined triangulation algorithm to the *ST_TINElement* ARRAY *elements* and constrained or modified by *maxsidelength*.

   c) Using the method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*, the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

8.6.3 ST_TINElements Methods

Purpose

Observe and mutate the ST_PrivateElements attribute of an ST_TIN value.

Definition

```
CREATE METHOD ST_TINElements()
   RETURNS ST_TINElement ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_TIN
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateElements
      END

CREATE METHOD ST_TINElements
   (elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements],
    triangulate INTEGER)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      DECLARE triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements];
      DECLARE acounter INTEGER;

      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN SELF;
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and elements.
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(SELF.ST_TINElements()) DO
         IF SELF.ST_SRID() <>
         SELF.ST_TINElements()[acounter].ST_ElementGeometry().ST_SRID THEN
            SIGNAL SQLSTATE '2FF10'
               SET MESSAGE_TEXT = 'mixed spatial reference systems';
         END IF;
         SET acounter = acounter + 1;
      END WHILE;
      -- If triangulate = 1, (re)triangulate the TIN surface
      triangles = SELF.ST_Patches();
      --
      -- Update triangles by triangulating – See Description
      --
      -- Return an ST_TIN value with the ST_PrivateControlPoints
      -- attribute set to controlpoints and, if triangulate = 1,
      -- with the ST_PrivatePatches attribute set to triangles.
      RETURN
         CASE
            WHEN triangulate = 1 THEN
               SELF.ST_PrivatePatches(triangles).
                  ST_PrivateElements(elements)
            ELSE
               SELF.ST_PrivateElements(elements);
   END
```

Description

1) The method *ST_TINElements()* has no input parameters.

2) For the null-call method *ST_TINElements()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateElements* attribute of SELF.

3) The method *ST_TINElements(ST_TINElement ARRAY, INTEGER)* takes the following input parameters:

   a) an *ST_TINElement* ARRAY value *elements*.

   b) an *INTEGER* value *triangulate*.

4) For the type-preserving method *ST_TINElements(ST_TINElement ARRAY, INTEGER)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) If the spatial reference system of SELF is not equal to the spatial reference system of all of the element geometries, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

   c) Case:

      i) If *triangulate* = 1, then:

         1) let *triangles = SELF.Patches().*

         2) let *elements = SELF.TINElements().*

         3) let *maxsidelength = SELF.MaxSideLength().*

         4) update *triangles* by applying the implementation-defined triangulation algorithm to the *ST_TINElements* ARRAY *elements* and constrained or modified by *maxsidelength.*

         5) return an *ST_TIN* value with the *ST_PrivatePatches* attribute set to *triangles* and the *ST_PrivateElements* attribute set to *elements*.

      ii) Otherwise, return an *ST_TIN* value with the *ST_PrivateElements* attribute set to *elements*.

5) It is implementation-defined which of the predefined TIN element types are supported, which additional TIN element types are supported, what type of *ST_Geometry* each requires, what behavior is to be expected during triangulation and what exceptions might be raised.

8.6.4 ST_MaxSideLength Methods

Purpose

Observe and mutate the ST_PrivateMaxSideLength attribute of an ST_TIN value.

Definition

```
CREATE METHOD ST_MaxSideLength()
   RETURNS DOUBLE PRECISION
   FOR ST_TIN
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateMaxSideLength
      END

CREATE METHOD ST_MaxSideLength
   (maxsidelength DOUBLE PRECISION,
    triangulate INTEGER)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      DECLARE triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements];

      IF maxsidelength IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      IF maxsidelength <= 0
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN SELF;
      END IF;
      -- If triangulate = 1, (re)triangulate the TIN surface
      triangles = SELF.ST_Patches();
      --
      -- Update triangles by triangulating – See Description
      --
      -- Return an ST_TIN value with the ST_PrivateMaxSideLength
      -- attribute set to maxsidelength and, if triangulate = 1,
      -- with the ST_PrivatePatches attribute set to triangles.
      RETURN
         CASE
            WHEN triangulate = 1 THEN
               SELF.ST_PrivatePatches(triangles).
                  ST_PrivateMaxSideLength(maxsidelength)
            ELSE
               SELF. ST_PrivateMaxSideLength(maxsidelength);
   END
```

Description

1) The method *ST_MaxSideLength()* has no input parameters.

2) For the null-call method *ST_MaxSideLength()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateMaxSideLength* attribute of SELF.

3) The method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) a DOUBLE PRECISION value *maxsidelength*.

   b) an INTEGER value *triangulate*.

4) For the type-preserving method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*:

   Case:

   a) If *maxsidelength* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *maxsidelength* is not greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   c) If SELF is the null value, then return the null value.

   d) Case:

      i) If *triangulate* = 1, then:

         1) let *triangles = SELF.Patches().*

         2) let *elements = SELF.TINElements().*

         3) update *triangles* by applying the implementation-defined triangulation algorithm to the *ST_TINElement* ARRAY *elements* and constrained or modified by *maxsidelength.*

         4) return an *ST_TIN* value with the *ST_PrivatePatches* attribute set to *triangles* and the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

      ii) Otherwise, return an *ST_TIN* value with the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

8.6.5 ST_TINTable Methods

Purpose

Observe and mutate the ST_TIN value in table format with point references.

Definition

```
CREATE METHOD ST_TINTable()
   RETURNS TABLE
   (item CHARACTER VARYING(30),
    ordernumber INTEGER,
    xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    triangle INTEGER ARRAY[3],
    visibility INTEGER,
    elementID INTEGER,
    elementtag CHARACTER VARYING(64),
    element INTEGER ARRAY[ST_MaxIntegerArrayElements])
   FOR ST_TIN
   BEGIN
      DECLARE triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements];
      DECLARE elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements];

      -- Get ST_TIN ARRAY values
      triangles = SELF.ST_Patches();
      elements = SELF.ST_TINElements();
      --
      -- See Description
      --
   END

CREATE METHOD ST_TINTable
   (tin_table_name DT1,
    item_column DT2,
    ordernumber_column DT2,
    xcoord_column DT2,
    ycoord_column DT2,
    zcoord_column DT2,
    triangle_column DT2,
    visibility_column DT2,
    elementID_column DT2,
    elementtag_column DT2,
    element_column DT2,
    maxsidelength DOUBLE PRECISION)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      DECLARE triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements];
      DECLARE elements ST_TINElement ARRAY[ST_MaxGeometryArrayElements];

      --
      -- See Description
      --
      RETURN SELF.ST_SRID(0).
         ST_Patches(triangles).
         ST_TINElements(elements,0).
         ST_MaxSideLength(maxsidelength,0)
   END
```


Definitional Rules

1) *ST_MaxIntegerArrayElements* is the implementation-defined maximum cardinality of an array of *INTEGER* elements.
2) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* elements.
3) *DT1* is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.
4) *DT2* is data type of variable-length character string with character set SQL_IDENTIFIER and maximum length not less than 128 characters.

Description

1) The method *ST_TINTable()* has no input parameters.
2) The method *TINTable()* returns the following value:
   a) a table value consisting of ten columns:
      i) a column item of type CHARACTER VARYING(30),
      ii) a column ordernumber of type INTEGER,
      iii) a column xcoord of type DOUBLE PRECISION,
      iv) a column *ycoord* of type DOUBLE PRECISION,
      v) a column *zcoord* of type DOUBLE PRECISION,
      vi) a column *triangle* of type INTEGER ARRAY[3],
      vii) a column *visibility* of type INTEGER, having values of -1 (hole), 0 (void), and 1 (visible),
      viii) a column *elementID* of type INTEGER,
      ix) a column *elementtag* of type CHARACTER VARYING(64),
      x) a column *element* of type INTEGER ARRAY[*ST_MaxIntegerArrayElements*].
3) For the null-call method *ST_TINTable()*:
   a) Let *triangles* be the *ST_Triangle* ARRAY returned by *SELF.PrivatePatches().*
   b) Let *elements* be the *ST_TINElement* ARRAY returned by *SELF.TINElements().*
   c) Let *points* be the *ST_Point* ARRAY created as follows
      i) For each *ST_Point* value in the *ST_MultiPoint* geometry of a TIN element in *elements* having *ST_ElementType()* = 'random points', add the *ST_Point* value to *points*,
      ii) For each *ST_Point* value in the *ST_MultiPoint* geometry of a TIN element in *elements* having *ST_ElementType()* = 'group spot', add the *ST_Point* value to *points*,
      iii) For each *ST_Point* value in the *ST_Polygon* geometry of a TIN element in *elements* having *ST_ElementType()* = 'boundary', add the *ST_Point* value to *points*,
      iv) For each *ST_Point* value in the *ST_LineString* geometry of a TIN element in *elements* having *ST_ElementType()* = 'breakline', add the *ST_Point* value to *points*,
      v) For each *ST_Point* value in the *ST_LineString* geometry of a TIN element in *elements* having *ST_ElementType()* = 'soft break', add the *ST_Point* value to *points*,
      vi) For each *ST_Point* value in the *ST_LineString* geometry of a TIN element in *elements* having *ST_ElementType()* = 'control contour', add the *ST_Point* value to *points*,
      vii) For each *ST_Point* value in the *ST_Polygon* geometry of a TIN element in *elements* having *ST_ElementType()* = 'break void', add the *ST_Point* value to *points*,
      viii) For each *ST_Point* value in the *ST_Polygon* geometry of a TIN element in *elements* having *ST_ElementType()* = 'drape void', add the *ST_Point* value to *points*,

ix) For each *ST_Point* value in the *ST_Polygon* geometry of a TIN element in *elements* having *ST_ElementType()* = 'void', add the *ST_Point* value to *points*,

x) For each *ST_Point* value in the *ST_Polygon* geometry of a TIN element in *elements* having *ST_ElementType()* = 'hole', add the *ST_Point* value to *points*,

xi) For each *ST_Point* value in the *ST_LineString* geometry of a TIN element in *elements* having *ST_ElementType()* = 'stop line', add the *ST_Point* value to *points*,

xii) Remove duplicate *ST_Point* values from points,

xiii) Reconstruct *points* by sorting the *ST_Point* values first by their *ST_Point.ST_X* value and then by their *ST_Point.ST_Y* value, both in increasing order.

d) Let *CP*, *T, V*, and E be *INTEGER* values.

e) For *CP* = 1 to *CARDINALITY(points)*:

i) Add a row to the TIN table with:

1) *item* = "point".

2) *ordernumber* = CP.

3) *xcoord = points[CP].ST_X*.

4) *ycoord = points[CP].ST_Y*.

5) *zcoord = points[CP].ST_Z*.

f) Output *triangles* as follows:

i) For *T* = 1 to *CARDINALITY(triangles)*:

1) Let *CP1* be an *INTEGER* value such that *points[CP1] = triangles[T].ST_Points[1]*.

2) Let *CP2* be an *INTEGER* value such that *points[CP2] = triangles[T].ST_Points[2]*.

3) Let *CP3* be an *INTEGER* value such that *points[CP3] = triangles[T].ST_Points[3]*.

4) Let *CPA* be an *INTEGER* value that is the smallest of *CP1*, *CP2* and *CP3*.

5) Let *CPB* be an *INTEGER* value that is the median of *CP1*, *CP2* and *CP3*.

6) Let *CPC* be an *INTEGER* value that is the largest of *CP1*, *CP2* and *CP3*.

7) For *E* = 1 to *CARDINALITY(elements)*:

Case

A) if (*elements[E].ElementType()* = 'break void' or *elements[E].ElementType()* = 'drape void' or *elements[E].ElementType()* = 'void') and *triangles[T].ST_Within(elements[E].ST_ElementGeometry())* = 1, then *V* = 0 (zero).

B) if *elements[E].ElementType()* = 'stop line' and *elements[E].ST_ElementGeometry()).ST_Crosses(triangles[T])* = 1, then *V* = 0 (zero).

C) if *elements[E].ElementType()* = 'hole' and *triangles[T].ST_Within(elements[E].ST_ElementGeometry())* = 1, then *V* = -1.

D) otherwise, *V* = 1 (one).

8) Add a row to the TIN table with:

A) *item* = "triangle".

B) *ordernumber = T.*

C) *visibility = V.*

D) *triangle* = ARRAY[*CPA CPB CPC*].

ii) Sort the triangles in the TIN Table by their *CPA*, then *CPB* and then *CPC* values in increasing order.

iii) Update the *ordernumber* of the sorted triangles in the TIN Table according to their sorted order position, with the triangle having the lowest *CPA, CPB, CPC* values getting an *ordernumber* equal to 1 (one) up to the triangle having the highest *CPA, CPB, CPC* values getting an *ordernumber* equal to *CARDINALITY(triangles)*.

g) Output *elements* as follows:

i) For *E* = 1 to *CARDINALITY(elements)*:

Case:

1) If *elements[E].ST_ElementGeometry().ST_GeometryType()* = 'ST_LineString':

A) Let *numepoints* be an *INTEGER* value equal to *TREAT(elements[E].ST_ElementGeometry() AS ST_LineString).ST_NumPoints*.

B) Let *CP4* be an *INTEGER* value such that *points[CP4]* = *TREAT(elements[E].ST_ElementGeometry() AS ST_LineString).ST_PointN[1]*.

C) Add a row to the TIN table with:

i) *item* = *elements[E].ElementType()*.

ii) *ordernumber* = *E*.

iii) *elementID* = *elements[E].ElementID()*.

iv) *elementtag* = *elements[E].ElementTag()*.

v) *element* = ARRAY[*CP4*].

D) Let *epointcounter* be an *INTEGER* value.

E) For *epointcounter* = 2 to *numepoints:*

i) Let *CP5* be an INTEGER value such that *controlpoints[CP5] = TREAT(elements[E].ST_ElementGeometry() AS ST_LineString).ST_PointN[epointcounter].*

ii) SET *element* = *element* || *CP5.*

2) If *elements[E].ST_ElementGeometry().ST_GeometryType()* = 'ST_Polygon':

A) Let *numepoints* be an *INTEGER* value equal to *TREAT(elements[E].ST_ElementGeometry() AS ST_Polygon).ST_ExteriorRing().ST_NumPoints*.

B) Let *CP4* be an *INTEGER* value such that *points[CP4]* = *TREAT(elements[E].ST_ElementGeometry() AS ST_Polygon).ST_ExteriorRing().ST_PointN[1]*.

C) Add a row to the TIN table with:

i) *item* = *elements[E].ElementType()*.

ii) *ordernumber* = *E*.

iii) *elementID* = *elements[E].ElementID()*.

iv) *elementtag* = *elements[E].ElementTag()*.

v) *element* = ARRAY[*CP4*].

D) Let *epointcounter* be an *INTEGER* value.

E) For *epointcounter* = 2 to *numepoints:*

i) Let *CP5* be an INTEGER value such that *controlpoints[CP5] = TREAT(elements[E].ST_ElementGeometry() AS ST_Polygon).ST_ExteriorRing().ST_PointN[epointcounter].*

ii) SET *element* = *element* || *CP5.*

F) If *TREAT(elements[E].ST_ElementGeometry() AS ST_Polygon).ST_NumInteriorRing() > 0*, then

i) Let *numholes* be an INTEGER value equal to *TREAT(elements[E].ST_ElementGeometry() AS ST_Polygon).ST_NumInteriorRing()*.

ii) Let *holecounter* be an INTEGER value.

iii) For *holecounter* = 1 to *numholes*:

1) Let *numipoints* be an INTEGER value equal to *elements[E]. TREAT(ST_ElementGeometry() AS ST_Polygon).ST_InteriorRingN(holecounter).ST_NumPoints().*

2) Let *CP6* be an INTEGER value such that *points[CP6] = elements[E]. TREAT(ST_ElementGeometry() AS ST_Polygon).ST_InteriorRingN(holecounter).ST_PointN[1]*.

3) Add a row to the TIN table with:

a) *item = elements[E].ElementType()* || ' hole'.

b) *ordernumber* = *E*.

c) *elementID = elements[E].ElementID().*

d) *elementtag = elements[E].ElementTag().*

e) *element* = ARRAY*[CP6]*.

4) Let *ipointcounter* be an INTEGER value.

5) For *ipointcounter* = 2 to *numipoints*:

a) Let *CP7* be an INTEGER value such that *controlpoints[CP7] = TREAT(elements[E].ST_ElementGeometry() AS ST_Polygon).ST_InteriorRingN(holecounter). ST_PointN[ipointcounter].*

b) SET *element* = *element* || *CP7*.

ii) Sort the elements in the TIN Table by their *item* value, then by the INTEGER values in their *element* ARRAY, from *element[1]* to *element[CARDINALITY(element)]*, all in increasing order.

iii) Update the *ordernumber* of the sorted elements in the TIN Table according to their sorted order position, with the element having the lowest *item* and INTEGER values getting an *ordernumber* equal to 1 (one) up to the element having the highest *item* and INTEGER values getting an *ordernumber* equal to *CARDINALITY(elements)*.

4) The method *ST_TINTable(DT1, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DOUBLE PRECISION)* takes the following input parameters:

a) a *DT1* value *tin_table_name*, which has the name of a referenced table.

i) The table consists of at least ten columns:

1) an item column of type CHARACTER VARYING(30) which identifies the type of TIN item contained in the row.

2) an ordernumber column of type INTEGER which identifies the position of the item value in sort order.

3) an xcoord column of type DOUBLE PRECISION which contains:

Case:

A) if *item* = 'point', then the x coordinate of the point.

B) otherwise, the NULL value.

4) a *ycoord* column of type DOUBLE PRECISION which contains:

   Case:

   A) if *item* = 'point', then the y coordinate of the point.

   B) otherwise, the NULL value.

5) a *zcoord* column of type DOUBLE PRECISION which contains:

   Case:

   A) if *item* = 'point', then the z coordinate of the point.

   B) otherwise, the NULL value.

6) a *triangle* column of type INTEGER ARRAY[3] which contains:

   Case:

   A) if *item* = 'triangle', then an array of the ordernumbers of the 3 points that define the triangle (the fourth point is always the same as the first so it is not included).

   B) otherwise, the NULL value.

7) a *visibility* column of type INTEGER which contains:

   Case:

   A) if *item* = 'triangle', then the *visibility* value of the triangle.

   B) otherwise, the NULL value.

8) an *elementID* column of type INTEGER which contains:

   Case:

   A) if *item* = 'element', then the *elementID* value of the element.

   B) otherwise, the NULL value.

9) an *elementtag* column of type CHARACTER VARYING(64) which contains:

   Case:

   A) if *item* = 'element', then the *elementtag* value of the element.

   B) otherwise, the NULL value.

10) an *element* column of type INTEGER ARRAY[*ST_MaxIntegerArrayElements*] which contains:

   Case:

   A) if *item* = 'element' and the element geometry is of type 'ST_LineString', then an array of the ordernumbers of the points that define the linestring.

   B) if *item* = 'element' and the element geometry is of type 'ST_Polygon', then an array of the ordernumbers of the points that define the exterior ring of the *ST_Polygon*.

   C) otherwise, the NULL value.

ii) Let *S* be the tin_table_name value.

iii) Let *V* be the character string that is the value of TRIM(BOTH ' ' FROM S).

iv) If *V* value does not conform to the Format and Syntax Rules of <table name> specified in ISO/IEC 9075-2, or the table specified by *paths_table_name* value does not exist in the system, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

b) a *DT2* value *item_column*, which has the name of the item column in the table specified by *tin_table_name*. If the column specified by *item_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

c) a *DT2* value *ordernumber_column*, which has the name of the ordernumber column in the table specified by *tin_table_name*. If the column specified by *ordernumber_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

d) a *DT2* value *xcoord_column*, which has the name of the xcoord column in the table specified by *tin_table_name*. If the column specified by *xcoord_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

e) a *DT2* value *ycoord_column*, which has the name of the *ycoord* column in the table specified by *tin_table_name*. If the column specified by *ycoord_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

f) a *DT2* value *zcoord_column*, which has the name of the *zcoord* column in the table specified by *tin_table_name*. If the column specified by *zcoord_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

g) a *DT2* value *triangle_column*, which has the name of the column for triangle in the table specified by *tin_table_name*. If the column specified by *triangle_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

h) a *DT2* value *visibility_column*, which has the name of the column for visibility in the table specified by *tin_table_name*. If the column specified by *visibility_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

i) a *DT2* value *elementID_column*, which has the name of the column for *elementID* in the table specified by *tin_table_name*. If the column specified by *elementID_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

j) a *DT2* value *elementtag_column*, which has the name of the column for *elementtag* in the table specified by *tin_table_name*. If the column specified by *elementtag_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

k) a *DT2* value *element_column*, which has the name of the column for element in the table specified by *tin_table_name*. If the column specified by *element_column* does not exist in the table specified by *tin_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

l) a DOUBLE PRECISION value *maxsidelength*, which specifies the maximum allowable triangle side length.

5) For the type-preserving method *ST_TINTable(DT1, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DT2, DOUBLE PRECISION)*:

a) Create a point ARRAY:

i) If there are not at least three UNIQUE rows in the table specified by *tin_table_name* having an *item_column* value of 'point', then an exception condition is raised: *SQL/MM Spatial exception – at least 3 points are required*.

ii) Let *points be an* empty *ST_Point* ARRAY,

iii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'point':

1) Let *X* be the DOUBLE PRECISION *xcoord_column* value.

2) Let *Y* be the DOUBLE PRECISION *ycoord_column* value.

3) Let *Z* be the DOUBLE PRECISION *zcoord_column* value.

4) Create an *ST_Point* value *CP* as *ST_Point(X, Y, Z)*.

5) SET *points* = *points* || *CP*.

b) Create a triangle ARRAY:

i) Let *triangles be an* empty *ST_Triangle* ARRAY.

ii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'triangle':

1) Let *Tarray* be the INTEGER ARRAY *triangle_column* value.

2) Let *TP1* be an *ST_Point* value equal to the *CP* control point *ST_Point* value created above for the row having an *item_column* value of "controlpoint" and an *index_column* value equal to *Tarray[1]*.

3) Let *TP2* be an *ST_Point* value equal to the *CP* control point *ST_Point* value created above for the row having an *item_column* value of "controlpoint" and an *index_column* value equal to *Tarray[2]*.

4) Let *TP3* be an *ST_Point* value equal to the *CP* control point *ST_Point* value created above for the row having an *item_column* value of "controlpoint" and an *index_column* value equal to *Tarray[3]*.

5) Let *TP4* be an *ST_Point* value equal to *TP1*.

6) Let *Tpointarray[TP1 TP2 TP3 TP4]* be an *ST_Point* ARRAY.

7) Create an *ST_Triangle* value *T* as *ST_Triangle(Tpointarray)*.

8) SET *triangles* = *triangles* || *T*.

c) Create an element ARRAY:

i) Let *elements be an* empty *ST_TINElement* ARRAY.

ii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'random points':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

B) SET *pointarray = pointarray* || *P.*

10) Create an *ST_MultiPoint* value *elementgeometry* as *ST_MultiPoint(pointarray)*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID. elementtag, elementgeometry)*,

12) SET *elements* = *elements* || *element*.

iii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'group spot':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.
2) Let *elementID* be the INTEGER *elementID_column* value.
3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.
4) Let *integerarray* be the INTEGER ARRAY *element_column* value.
5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.
6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.
7) Let *pointarray[P1]* be an *ST_Point* ARRAY.
8) Let *pointcounter* be an INTEGER value.
9) For *pointcounter* = 2 to *numpoints:*
   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.
   B) SET *pointarray* = *pointarray* || *P.*
10) Create an *ST_MultiPoint* value *elementgeometry* as *ST_MultiPoint(pointarray)*.
11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.
12) SET *elements* = *elements* || *element*.

iv) For the UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'boundary':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.
2) Let *elementID* be the INTEGER *elementID_column* value.
3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.
4) Let *integerarray* be the INTEGER ARRAY *element_column* value.
5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.
6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.
7) Let *pointarray[P1]* be an *ST_Point* ARRAY.
8) Let *pointcounter* be an INTEGER value.
9) For *pointcounter* = 2 to *numpoints:*
   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.
   B) SET *pointarray* = *pointarray* || *P.*
10) Create an *ST_Polygon* value *elementgeometry* as *ST_Polygon(ST_LineString(pointarray))*.
11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.
12) SET *elements* = *elements* || *element*.

v) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'breakline':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray = pointarray* || *P.*

10) Create an *ST_Linestring* value *elementgeometry* as *ST_LineString(pointarray)*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

vi) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'soft break':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray = pointarray* || *P.*

10) Create an *ST_Linestring* value *elementgeometry* as *ST_LineString(pointarray)*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

vii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'control contour':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray* = *pointarray* || *P.*

10) Create an *ST_Linestring* value *elementgeometry* as *ST_LineString(pointarray)*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

viii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'break void':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray* = *pointarray* || *P.*

10) Create an *ST_Polygon* value *elementgeometry* as *ST_Polygon(ST_LineString(pointarray))*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

ix) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'break void hole':

1) Let *holeintegerarray* be the INTEGER ARRAY *element_column* value.

2) Let *numholepoints* be an INTEGER value equal to *CARDINALITY(holeintegerarray)*.

3) Let *HP1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *holeintegerarray[1]*.

4) Let *holepointarray[HP1]* be an *ST_Point* ARRAY.

5) Let *holepointcounter* be an INTEGER value.

6) For *holepointcounter* = 2 to *numholepoints*:

   A) Let *HP* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *holeintegerarray[holepointcounter]*.

   B) SET *holepointarray* = *holepointarray* || *HP*.

7) Create an *ST_LineString* value *holering* as *ST_LineString(holepointarray)*.

8) Let *HE* be an INTEGER value such that *elements[HE].ST_ElementType* = 'break void' and *holering.ST_Within(elements[HE].ST_ElementGeometry().ST_ExteriorRing()) = 1* and *ST_Area(elements[HE].ST_ElementGeometry().ST_ExteriorRing().ST_Envelope)* is the smallest area of all break void elements created above.

9) Let *holesarray* be the *ST_Curve* ARRAY equal to *elements[HE].ST_ElementGeometry().ST_InteriorRings()*.

10) SET *holesarray* = *holesarray* || *holering*.

11) Add the holes found so far to the break void element using *elements[HE].ST_ElementGeometry.ST_InteriorRings(holesarray)*.

x) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'drape void':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray* = *pointarray* || *P.*

10) Create an *ST_Polygon* value *elementgeometry* as *ST_Polygon(ST_LineString(pointarray))*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

xi) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'drape void hole':

1) Let *holeintegerarray* be the INTEGER ARRAY *element_column* value.

2) Let *numholepoints* be an INTEGER value equal to *CARDINALITY(holeintegerarray)*.

3) Let *HP1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *holeintegerarray[1]*.

4) Let *holepointarray[HP1]* be an *ST_Point* ARRAY.

5) Let *holepointcounter* be an INTEGER value.

6) For *holepointcounter = 2* to *numholepoints*:

   A) Let *HP* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *holeintegerarray[holepointcounter]*.

   B) SET *holepointarray = holepointarray* || *HP*.

7) Create an *ST_LineString* value *holering* as *ST_LineString(holepointarray)*.

8) Let *HE* be an INTEGER value such that *elements[HE].ST_ElementType* = 'drape void' and *holering.ST_Within(elements[HE].ST_ElementGeometry().ST_ExteriorRing()) = 1* and *ST_Area(elements[HE].ST_ElementGeometry().ST_ExteriorRing().ST_Envelope)* is the smallest area of all drape void elements created above.

9) Let *holesarray* be the *ST_Curve* ARRAY equal to *elements[HE].ST_ElementGeometry().ST_InteriorRings()*.

10) SET *holesarray = holesarray* || *holering*.

11) Add the holes found so far to the drape void element using *elements[HE].ST_ElementGeometry.ST_InteriorRings(holesarray)*.

xii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'void':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray = pointarray* || *P.*

10) Create an *ST_Polygon* value *elementgeometry* as *ST_Polygon(ST_LineString(pointarray))*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

xiii) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'void hole':

1) Let *holeintegerarray* be the INTEGER ARRAY *element_column* value.

2) Let *numholepoints* be an INTEGER value equal to *CARDINALITY(holeintegerarray)*.

3) Let *HP1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *holeintegerarray[1]*.

4) Let *holepointarray[HP1]* be an *ST_Point* ARRAY.

5) Let *holepointcounter* be an INTEGER value.

6) For *holepointcounter = 2* to *numholepoints*:

   A) Let *HP* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *holeintegerarray[holepointcounter]*.

   B) SET *holepointarray = holepointarray* || *HP*.

7) Create an *ST_LineString* value *holering* as *ST_LineString(holepointarray)*.

8) Let *HE* be an INTEGER value such that *elements[HE].ST_ElementType* = 'void' and *holering.ST_Within(elements[HE].ST_ElementGeometry().ST_ExteriorRing()) = 1* and *ST_Area(elements[HE].ST_ElementGeometry().ST_ExteriorRing().ST_Envelope)* is the smallest area of all void elements created above.

9) Let holesarray be the *ST_Curve* ARRAY equal to *elements[HE].ST_ElementGeometry().ST_InteriorRings()*.

10) SET *holesarray = holesarray* || *holering*.

11) Add the holes found so far to the void element using *elements[HE].ST_ElementGeometry.ST_InteriorRings(holesarray)*.

xiv) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'hole':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray = pointarray* || *P.*

10) Create an *ST_Polygon* value *elementgeometry* as *ST_Polygon(ST_LineString(pointarray))*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements* = *elements* || *element*.

xv) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'hole hole':

1) Let *holeintegerarray* be the INTEGER ARRAY *element_column value*.

2) Let *numholepoints* be an INTEGER value equal to *CARDINALITY(holeintegerarray)*.

3) Let *HP1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *holeintegerarray[1]*.

4) Let *holepointarray[HP1]* be an *ST_Point* ARRAY.

5) Let *holepointcounter* be an INTEGER value.

6) For *holepointcounter* = 2 to *numholepoints*:

   A) Let *HP* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *holeintegerarray[holepointcounter]*.

   B) SET *holepointarray = holepointarray* || *HP*.

7) Create an *ST_LineString* value *holering* as *ST_LineString(holepointarray)*.

8) Let *HE* be an INTEGER value such that *elements[HE].ST_ElementType* = 'hole' and *holering.ST_Within(elements[HE].ST_ElementGeometry().ST_ExteriorRing()) = 1* and *ST_Area(elements[HE].ST_ElementGeometry().ST_ExteriorRing().ST_Envelope)* is the smallest area of all hole elements created above.

9) Let *holesarray* be the *ST_Curve* ARRAY equal to *elements[HE].ST_ElementGeometry().ST_InteriorRings()*.

10) SET *holesarray = holesarray* || *holering*.

11) Add the holes found so far to the hole element using *elements[HE].ST_ElementGeometry.ST_InteriorRings(holesarray)*.

xvi) For each UNIQUE row in the table specified by *tin_table_name* having an *item_column* value of 'stop line':

1) Let *elementtype* be the CHARACTER VARYING *item_column* value.

2) Let *elementID* be the INTEGER *elementID_column* value.

3) Let *elementtag* be the CHARACTER VARYING *elementtag_column* value.

4) Let *integerarray* be the INTEGER ARRAY *element_column* value.

5) Let *numpoints* be an INTEGER value equal to *CARDINALITY(integerarray)*.

6) Let *P1* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *ordernumber_column* value equal to *integerarray[1]*.

7) Let *pointarray[P1]* be an *ST_Point* ARRAY.

8) Let *pointcounter* be an INTEGER value.

9) For *pointcounter* = 2 to *numpoints:*

   A) Let *P* be an *ST_Point* value equal to the *CP* point *ST_Point* value created above for the row having an *item_column* value of 'point' and an *index_column* value equal to *integerarray[pointcounter]*.

   B) SET *pointarray = pointarray* || *P.*

10) Create an *ST_Linestring* value *elementgeometry* as *ST_LineString(pointarray)*.

11) Create an *element* value *element* as *ST_TINElement(elementtype, elementID, elementtag, elementgeometry)*.

12) SET *elements = elements* || *element*.

d) If *CARDINALITY(triangles)* = 0, create *ST_Triangles* by applying the implementation-defined triangulation algorithm to the *ST_Point* ARRAY *points* and constrained or modified by *elements* and *maxsidelength*.

e) Return an *ST_TIN* value with:

i) The spatial reference system identifier set to 0 (zero).

ii) Using the method *ST_Patches(ST_Triangle* ARRAY*)*:

1) the *ST_PrivateDimension* attribute set to 2.

2) the *ST_PrivateCoordinateDimension* attribute set to the value expression *ST_GetCoordDim(triangles)*.

3) the *ST_PrivateIs3D* attribute set to 1 (one).

4) the *ST_PrivateIsMeasured* attribute set to the value expression *ST_GetIsMeasured(triangles).*

5) the *ST_PrivatePatches* attribute set to *triangles*.

iii) Using the method *ST_TINElements(ST_TINElement* ARRAY*, INTEGER)*, the *ST_PrivateElements* attribute set to *elements*.

iv) Using the method *ST_MaxSideLength(DOUBLE PRECISION, INTEGER)*, the *ST_PrivateMaxSideLength* attribute set to *maxsidelength*.

8.6.6 ST_Clip Method

Purpose

Returns that part of an ST_TIN value that is within the clipping boundary.

Definition

```
CREATE METHOD ST_Clip
   (clippolygon ST_Polygon)
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      --
      --  See Description
      --
   END
```

Description

1) The method *ST_Clip(ST_Polygon)* takes the following input parameters:
   a) an *ST_Polygon* value *clippolygon*.
2) For the type-preserving method *ST_Clip(ST_Polygon)*:
   a) The ST_Polygon value *clippolygon* becomes the new boundary for *SELF*.
   b) That part of *SELF* which is outside of this boundary is removed.

8.6.7 ST_Patches Methods

Purpose

Observe and mutate the ST_PrivatePatches attribute of an ST_TIN value.

Definition

```
CREATE METHOD ST_Patches()
   RETURNS ST_Triangle ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_TIN
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivatePatches
      END

CREATE METHOD ST_Patches
   (triangles ST_Triangle ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_TIN
   FOR ST_TIN
   BEGIN
      -- If triangles is not an ST_Triangle ARRAY, then raise an exception
      IF triangles IS NOT OF (ST_Triangle ARRAY) THEN
         SIGNAL SQLSTATE '2FF67'
            SET MESSAGE_TEXT = 'polygon value is not a triangle value';
      END IF;
      RETURN (SELF AS ST_PolyhdrlSurface).ST_Patches(triangles);
   END
```

Description

1) The method *ST_Patches()* has no input parameters.

2) For the null-call method *ST_Patches()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivatePatches* attribute of SELF.

3) The method *ST_Patches(ST_Triangle ARRAY)* takes the following input parameters:

   a) an *ST_Triangle* value *apolygonarray*.

4) For the type-preserving method *ST_Patches(ST_Triangle ARRAY)*:

   Case:

   a) If *triangles* is not an ST_Triangle ARRAY value, then an exception condition is raised: *SQL/MM Spatial exception – polygon value is not a triangle value*.

   b) Otherwise, return an *ST_TIN* value as a result of the value expression: *(SELF AS ST_PolyhdrlSurface).ST_Patches(triangles).*

8.6.8 ST_TINFromText Functions

Purpose

Return an ST_TIN value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_TIN value.

Definition

```
CREATE FUNCTION ST_TINFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_TINFromText(awkt, 0)

CREATE FUNCTION ST_TINFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_TINFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_TINFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_TINFromText(awkt, 0)*.
3) The function *ST_TINFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_TINFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_TIN* value.

      If *awkt* is not producible in the BNF for <tin text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_TIN)*.

8.6.9 ST_TINFromWKB Functions

Purpose

Return an ST_TIN value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_TIN value.

Definition

```
CREATE FUNCTION ST_TINFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_TINFromWKB(awkb, 0)

CREATE FUNCTION ST_TINFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_TINFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_TINFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_TINFromWKB(awkb, 0)*.
3) The function *ST_TINFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_TINFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_TIN* value.

      If *awkb* is not producible in the BNF for <tin binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_TIN)*.

8.6.10 ST_TINFromGML Functions

Purpose

Return an ST_TIN value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML 3.2.1 or 3.3 representation of an ST_TIN value.

Definition

```
CREATE FUNCTION ST_TINFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_TINFromGML(agml, 0)

CREATE FUNCTION ST_TINFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_TIN
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_TINFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_TINFromGML*(CHARACTER LARGE OBJECT) returns the result of the value expression: *ST_TINFromGML(agml, 0)*.
3) The function *ST_TINFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_TINFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a TIN XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_TIN)*.

8.7 ST_CompoundSurface Type and Routines

8.7.1 ST_CompoundSurface Type

Purpose

The general notion of a compound surface is a collection of surfaces that join in pairs on common boundary curves and which, when considered as a whole, form a single surface. The contributing surface types include all subtypes of ST_Surface.

Definition

```
CREATE TYPE ST_CompoundSurface
   UNDER ST_Surface
   AS (
      ST_PrivateSurfaces ST_Surface
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_CompoundSurface
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_CompoundSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CompoundSurface
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_CompoundSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CompoundSurface
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_CompoundSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_CompoundSurface
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_CompoundSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_CompoundSurface(asurface ST_Surface)
   RETURNS ST_CompoundSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CompoundSurface
   (asurface ST_Surface,
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CompoundSurface
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_CompoundSurface
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Surfaces()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Surfaces
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumSurfaces()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_SurfaceN
   (aposition INTEGER)
   RETURNS ST_Surface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivateSurfaces* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateSurfaces*.

Description

1) The *ST_CompoundSurface* type provides for public use:
   a) a method *ST_CompoundSurface(CHARACTER LARGE OBJECT)*,
   b) a method *ST_CompoundSurface(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_CompoundSurface(BINARY LARGE OBJECT)*,
   d) a method *ST_CompoundSurface(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_CompoundSurface(ST_Surface)*,
   f) a method *ST_CompoundSurface(ST_Surface, INTEGER)*,
   g) a method *ST_CompoundSurface(ST_Surface ARRAY)*,
   h) a method *ST_CompoundSurface(ST_Surface ARRAY, INTEGER)*,
   i) a method *ST_Surfaces()*,
   j) a method *ST_Surfaces(ST_Surface ARRAY)*,
   k) a method *ST_NumSurfaces()*,
   l) a method *ST_SurfaceN(INTEGER)*,
   m) a function *ST_CompSurfFromTxt(CHARACTER LARGE OBJECT)*,
   n) a function *ST_CompSurfFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,
   o) a function *ST_CompSurfFromWKB(BINARY LARGE OBJECT)*,
   p) a function *ST_CompSurfFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   q) a function *ST_CompSurfFromGML(CHARACTER LARGE OBJECT)*,
   r) a function *ST_CompSurfFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The *ST_PrivateSurfaces* attribute contains a collection of *ST_Surface* values.
3) If each *ST_Surface* value in the *ST_PrivateSurfaces* attribute is well formed, then the *ST_CompoundSurface* value is well formed.
4) All the *ST_Surface* values in the *ST_PrivateSurfaces* attribute are in the same spatial reference system as the *ST_CompoundSurface* value.
5) The *ST_PrivateSurfaces* attribute shall not be the null value. The elements in the *ST_PrivateSurfaces* attribute shall not be the null value.

6) The coordinate dimension of an *ST_CompoundSurface* value is equal to the coordinate dimension of its *ST_Surface* values.
7) An *ST_CompoundSurface* value consists of one or more surfaces joined in pairs on common boundary curves and which, when considered as a whole, form a single surface. The contributing surface types include all subtypes of *ST_Surface*.
8) If an *ST_CompoundSurface* value is simple and closed, then it is considered a shell.
9) An *ST_CompoundSurface* value returned by the constructor function corresponds to the empty set.
10) An *ST_CompoundSurface* value with the cardinality of the attribute *ST_PrivateSurfaces* equal to 0 (zero) corresponds to the empty set.

8.7.2 ST_CompoundSurface Methods

Purpose

Return an ST_CompoundSurface value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Surface values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN NEW ST_CompoundSurface(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN NEW ST_CompoundSurface(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN ST_CompSurfFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (asurface ST_Surface)
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN SELF.ST_SRID(0).ST_Surfaces(ARRAY[asurface])

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (asurface ST_Surface,
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN SELF.ST_SRID(ansrid).ST_Surfaces(ARRAY[asurface])

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN SELF.ST_SRID(0).ST_Surfaces(asurfacearray)

CREATE CONSTRUCTOR METHOD ST_CompoundSurface
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   RETURN SELF.ST_SRID(ansrid).ST_Surfaces(asurfacearray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_CompoundSurface(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_CompoundSurface(awktorgml, 0)*.
3) The method *ST_CompoundSurface(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(CHARACTER LARGE OBJECT, INTEGER)*:

   Case;
   a) If *awktorgml* contains a CompositeSurface XML element in the GML representation, then return the result of the value expression: *ST_CompSurfFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_CompSurfFromTxt(awktorgml, ansrid)*.
5) The method *ST_CompoundSurface(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_CompoundSurface(awkb, 0)*.
7) The method *ST_CompoundSurface(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
8) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_CompSurfFromWKB(awkb, ansrid).*
9) The method *ST_CompoundSurface(ST_Surface)* takes the following input parameters:
   b) an *ST_Surface* value *asurface*.
10) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(ST_Surface)* returns an *ST_CompoundSurface* value with:
    a) The spatial reference system identifier set to 0 (zero).
    b) Let *asurfacearray* be an *ST_Surface* ARRAY containing a single element, *asurface*.
    c) Using the method *ST_Surfaces(ST_Surface ARRAY):*
       i) the *ST_PrivateDimension* attribute set to 2.
       ii) the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(asurfacearray).*

iii) the *ST_PrivateIs3D* attribute set to *ST_GetIs3D(asurfacearray)*.

iv) the *ST_PrivateIsMeasured* attribute set to *ST_GetIsMeasured(asurfacearray)*.

v) the *ST_PrivateSurfaces* attribute set to *asurfacearray*.

11) The method *ST_CompoundSurface(ST_Surface, INTEGER)* takes the following input parameters:

a) an *ST_Surface* value *asurface*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(ST_Surface, INTEGER)* returns an *ST_CompoundSurface* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Let *asurfacearray* be an *ST_Surface* ARRAY containing a single element, *asurface*.

c) Using the method *ST_Surfaces(ST_Surface ARRAY):*

i) the *ST_PrivateDimension* attribute set to 2.

ii) the ST_PrivateCoordinateDimension attribute set to *ST_GetCoordDim(asurfacearray)*.

iii) the *ST_PrivateIs3D* attribute set to *ST_GetIs3D(asurfacearray)*.

iv) the *ST_PrivateIsMeasured* attribute set to *ST_GetIsMeasured(asurfacearray)*.

v) the *ST_PrivateSurfaces* attribute set to *asurfacearray*.

13) The method *ST_CompoundSurface(ST_Surface ARRAY)* takes the following input parameters:

a) an *ST_Surface* ARRAY value *asurfacearray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(ST_Surface ARRAY)* returns an *ST_CompoundSurface* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_Surfaces(ST_Surface ARRAY):*

i) the *ST_PrivateDimension* attribute set to 2.

ii) the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(asurfacearray)*.

iii) the *ST_PrivateIs3D* attribute set to *ST_GetIs3D(asurfacearray)*.

iv) the *ST_PrivateIsMeasured* attribute set to *ST_GetIsMeasured(asurfacearray)*.

v) the *ST_PrivateSurfaces* attribute set to *asurfacearray*.

15) The method *ST_CompoundSurface(ST_Surface ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Surface* ARRAY value *asurfacearray*,

b) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_CompoundSurface(ST_Surface ARRAY, INTEGER)* returns an *ST_CompoundSurface* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Surfaces(ST_Surface ARRAY):*

i) the *ST_PrivateDimension* attribute set to 2.

ii) the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(asurfacearray)*.

iii) the *ST_PrivateIs3D* attribute set to *ST_GetIs3D(asurfacearray)*.

iv) the *ST_PrivateIsMeasured* attribute set to *ST_GetIsMeasured(asurfacearray)*.

v) the *ST_PrivateSurfaces* attribute set to *asurfacearray*.

8.7.3 ST_Surfaces Methods

Purpose

Observe and mutate the ST_PrivateSurfaces attribute of an ST_CompoundSurface value.

Definition

```
CREATE METHOD ST_Surfaces()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_CompoundSurface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateSurfaces
      END

CREATE METHOD ST_Surfaces
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundSurface
   FOR ST_CompoundSurface
   BEGIN
      DECLARE counter INTEGER;

      -- If asurfacearray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(asurfacearray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_CompoundSurface);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and asurfacearray.
      IF (CARDINALITY(asurfacearray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(asurfacearray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- If any surface is not contiguous (share part of its boundary)
      -- with at least one other surface, then raise an exception
      SET counter = 2;
      WHILE counter <= CARDINALITY(asurfacearray) DO
            IF asurfacearray[counter].ST_Intersection
               (asurfacearray[counter-1]).ST_Dimension() < > 1 THEN
            SIGNAL SQLSTATE '2FF85'
               SET MESSAGE_TEXT = 'non-contiguous surfaces';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      -- If SELF is the null value, then return the null value. Otherwise,
      -- return an ST_CompoundSurface value with the ST_PrivateSurfaces
      -- attribute set to asurfacearray.
      RETURN
         SELF.ST_PrivateDimension(2).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(asurfacearray)).
            ST_PrivateIs3D(ST_GetIs3D(asurfacearray)).
            ST_PrivateIsMeasured(ST_GetIsMeasured(asurfacearray)).
            ST_PrivateSurfaces(asurfacearray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Surfaces()* has no input parameters.
2) For the null-call method *ST_Surfaces()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateSurfaces* attribute of SELF.
3) The method *ST_Surfaces(ST_Surface ARRAY)* takes the following input parameters:
   a) an *ST_Surface* ARRAY value *asurfacearray*.
4) For the type-preserving method *ST_Surfaces(ST_Surface ARRAY)*:
   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *asurfacearray* is the null value or contains null elements.
   b) Case:
      i) If SELF is the null value, then return the null value.
      ii) If the cardinality of *asurfacearray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID(asurfacearray)*, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.
      iii) If any *ST_Surface* value is not contiguous (shares part of its boundary) with at least one other *ST_Surface* value, then an exception condition is raised: *SQL/MM Spatial exception – non-contiguous surfaces*.
      iv) Otherwise, return an *ST_CompoundSurface* value with:
         1) The dimension set to 1 (one).
         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(asurfacearray)*.
         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(asurfacearray)*.
         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(asurfacearray)*.
         5) The *ST_PrivateSurfaces* attribute set to *asurfacearray*.

8.7.4 ST_NumSurfaces Method

Purpose

Return the cardinality of the ST_PrivateSurfaces attribute of an ST_CompoundSurface value.

Definition

```
CREATE METHOD ST_NumSurfaces()
   RETURNS INTEGER
   FOR ST_CompoundSurface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivateSurfaces)
      END
```

Description

1) The method *ST_NumSurfaces()* has no input parameters.
2) For the null-call method *ST_NumSurfaces()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the cardinality of the *ST_PrivateSurfaces* attribute.

8.7.5 ST_SurfaceN Method

Purpose

Return the specified element in the ST_PrivateSurfaces attribute of an ST_CompoundSurface value.

Definition

```
CREATE METHOD ST_SurfaceN
   (aposition INTEGER)
   RETURNS ST_Surface
   FOR ST_CompoundSurface
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Surface);
      END IF;
      IF aposition < 1 OR
         aposition > CARDINALITY(SELF.ST_PrivateSurfaces) THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Surface);
         END;
      END IF;
      RETURN SELF.ST_PrivateSurfaces[aposition];
   END
```

Description

1) The method *ST_SurfaceN(INTEGER)* takes the following input parameters:

   a) an INTEGER value *aposition*.

2) For the null-call method *ST_SurfaceN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If *aposition* is less than 1 (one) or greater than the cardinality of the *ST_PrivateSurfaces* attribute, then:

      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.

      ii) Return the null value.

   c) Otherwise, return an *ST_Surface* value at element *aposition* in the *ST_PrivateSurfaces* attribute of SELF.

8.7.6 ST_CompSurfFromTxt Functions

Purpose

Return an ST_CompoundSurface value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CompoundSurface value.

Definition

```
CREATE FUNCTION ST_CompSurfFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CompSurfFromTxt(awkt, 0)

CREATE FUNCTION ST_CompSurfFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_CompSurfFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_CompSurfFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CompSurfFromTxt(awkt, 0)*.
3) The function *ST_CompSurfFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CompSurfFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_CompoundSurface* value.

      If *awkt* is not producible in the BNF for <compoundsurface text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_CompoundSurface)*.

8.7.7 ST_CompSurfFromWKB Functions

Purpose

Return an ST_CompoundSurface value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CompoundSurface value.

Definition

```
CREATE FUNCTION ST_CompSurfFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CompSurfFromWKB(awkb, 0)

CREATE FUNCTION ST_CompSurfFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_CompSurfFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_CompSurfFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_CompSurfFromWKB(awkb, 0)*.
3) The function *ST_CompSurfFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CompSurfFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkb* is the well-known binary representation of an *ST_CompoundSurface* value.

      If *awkb* is not producible in the BNF for <compoundsurface binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_CompoundSurface)*.

8.7.8 ST_CompSurfFromGML Functions

Purpose

Return an ST_CompoundSurface value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_CompoundSurface value.

Definition

```
CREATE FUNCTION ST_CompSurfFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_CompSurfFromGML(agml, 0)

CREATE FUNCTION ST_CompSurfFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_CompoundSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_CompSurfFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_CompSurfFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_CompSurfFromGML(agml, 0)*.
3) The function *ST_CompSurfFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_CompSurfFromGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a CompositeSurface XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: TREAT(ST_GeomFromGML(agml, ansrid) AS ST_CompoundSurface).

9 Solid Types

9.1 ST_Solid Type and Routines

9.1.1 ST_Solid Type

Purpose

The ST_Solid type is a supertype for 3-dimensional geometry types.

Definition

```
CREATE TYPE ST_Solid
   UNDER ST_Geometry
   NOT INSTANTIABLE
   NOT FINAL

   METHOD ST_3DSurfaceArea()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DSurfaceArea
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DVolume()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DVolume
      (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DCentroid()
      RETURNS ST_Point
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_3DPointOnSolid()
      RETURNS ST_Point
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The *ST_Solid* type provides for public use:
   a) a method *ST_3DSurfaceArea()*,
   b) a method *ST_3DSurfaceArea(CHARACTER VARYING)*,
   c) a method *ST_3DVolume()*,
   d) a method *ST_3DVolume(CHARACTER VARYING)*,
   e) a method *ST_3DCentroid()*,
   f) a method *ST_3DPointOnSolid()*.
2) An *ST_Solid* value is a 3-dimensional *ST_Geometry* value representing the continuous image of a region of Euclidean 3 space.
3) The dimension of an *ST_Solid* value is 3.
4) *ST_Solid* values shall have z coordinate values, so *SELF.ST_Is3D()* is 1 (one).
5) *ST_Solid* values shall not have m coordinate values, so *SELF.ST_IsMeasured()* is 0 (zero).
6) Because *ST_Solid* values shall have z coordinate values but not m coordinate values, *SELF.ST_CoordDim()* is 3.

9.1.2 ST_3DSurfaceArea Methods

Purpose

Return the sum of the surface areas of all of the boundary components of an ST_Solid value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DSurfaceArea()
   RETURNS DOUBLE PRECISION
   FOR ST_Solid
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_3DSurfaceArea
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Solid
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DSurfaceArea()* has no input parameters.
2) For the null-call method *ST_3DSurfaceArea()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined surface area of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *3DSurfaceArea()* is in the linear unit of measure identified by <linear unit> squared.
      ii) Otherwise, the value returned by *ST_3DSurfaceArea()* is in an implementation-defined unit of measure.
3) The method *ST_3DSurfaceArea(CHARACTER VARYING)* takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_3DSurfaceArea(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the area of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
   d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the implementation-defined area of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

9.1.3 ST_3DVolume Methods

Purpose

Return the volume measurement of an ST_Solid value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DVolume()
   RETURNS DOUBLE PRECISION
   FOR ST_Solid
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_3DVolume
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_Solid
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DVolume()* has no input parameters.
2) For the null-call method *ST_3DVolume()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the implementation-defined volume of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.
   b) Case:
      i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DVolume()* is in the linear unit of measure identified by <linear unit> squared.
      ii) Otherwise, the value returned by *ST_3DVolume()* is in an implementation-defined unit of measure.
3) The method *ST_3DVolume(CHARACTER VARYING)* \takes the following input parameter:
   a) a CHARACTER VARYING value *aunit*.
4) For the null-call method *ST_3DVolume(CHARACTER VARYING)*:
   a) The values for *aunit* shall be a supported <unit name>.
   b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.
   c) If the unit specified by *aunit* is not supported by the implementation to compute the volume of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

   i) If SELF is an empty set, then return the null value.

   ii) Otherwise, return the implementation-defined volume of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.

e) The returned value is in the units indicated by *aunit*.

9.1.4 ST_3DCentroid Method

Purpose

Return the ST_Point value that is the mathematical centroid of the ST_Solid value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DCentroid()
   RETURNS ST_Point
   FOR ST_Solid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
      -- ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_3DCentroid()* has no input parameters.

2) For the null-call method *ST_3DCentroid()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Return the mathematical centroid of the *ST_Solid* value. The result is not guaranteed to spatially intersect the *ST_Solid* value.

      ii) The *ST_Point* value does not include an m coordinate value.

      iii) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

9.1.5 ST_3DPointOnSolid Method

Purpose

Return an ST_Point value guaranteed to spatially intersect the ST_Solid value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DPointOnSolid()
   RETURNS ST_Point
   FOR ST_Solid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
      -- ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_3DPointOnSolid()* has no input parameters.
2) For the null-call method *ST_3DPointOnSolid()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Return an *ST_Point* value guaranteed to spatially 3D intersect the *ST_Solid* value.
      ii) The *ST_Point* value does not include the m coordinate value.
      iii) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

9.2 ST_BRepSolid Type and Routines

9.2.1 ST_BRepSolid Type

Purpose

The ST_BRepSolid type is a subtype of the ST_Solid. The ST_BRepSolid type is instantiable. An ST_BRepSolid value is a 3-dimensional geometry that consists of a single connected interior that is associated with one exterior shell and zero or more interior shells.

Definition

```
CREATE TYPE ST_BRepSolid
   UNDER ST_Solid
   AS (
      ST_PrivateExteriorShell ST_Surface,
      ST_PrivateInteriorShells ST_Surface,
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_BRepSolid
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_BRepSolid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_BRepSolid
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_BRepSolid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_BRepSolid
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_BRepSolid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_BRepSolid
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_BRepSolid
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```



```
CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface)
   RETURNS ST_BRepSolid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface,
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface,
    asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_BRepSolid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface,
    asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ExteriorShell()
   RETURNS ST_Surface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ExteriorShell(asurface ST_Surface)
   RETURNS ST_BRepSolid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_InteriorShells()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_InteriorShells
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_BRepSolid
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_NumIntShells()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_InteriorShellN
   (aposition INTEGER)
   RETURNS ST_Surface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivateExteriorShell* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateExteriorShell*.
5) The attribute *ST_PrivateInteriorShells* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateInteriorShells*.

Description

1) The *ST_BRepSolid* type provides for public use:
   a) a method *ST_BRepSolid(CHARACTER LARGE OBJECT)*,
   b) a method *ST_BRepSolid(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_BRepSolid(BINARY LARGE OBJECT)*,
   d) a method *ST_BRepSolid (BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_BRepSolid (ST_Surface)*,
   f) a method *ST_BRepSolid (ST_Surface, INTEGER)*,
   g) a method *ST_BRepSolid (ST_Surface, ST_Surface ARRAY)*,
   h) a method *ST_BRepSolid (ST_Surface, ST_Surface ARRAY, INTEGER)*,
   i) a method *ST_ExteriorShell()*,
   j) a method *ST_ExteriorShell(ST_Surface)*,
   k) a method *ST_InteriorShells()*,
   l) a method *ST_InteriorShells(ST_Surface ARRAY)*,

m) a method *ST_NumIntShells()*,

n) a method *ST_InteriorShellN(INTEGER)*,

o) a function *ST_BRepFromText(CHARACTER LARGE OBJECT)*,

p) a function *ST_BRepFromText(CHARACTER LARGE OBJECT, INTEGER)*,

q) a function *ST_BRepFromWKB(BINARY LARGE OBJECT)*,

r) a function *ST_BRepFromWKB(BINARY LARGE OBJECT, INTEGER)*,

s) a function *ST_BRepFromGML(CHARACTER LARGE OBJECT)*,

t) a function *ST_BRepFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The *ST_PrivateExteriorShell* attribute is an *ST_Surface* value that is a shell.

3) The *ST_PrivateInteriorShells* attribute is a collection of *ST_Surface* values. Each *ST_Surface* value in the collection is a shell.

4) The *ST_PrivateExteriorShell* attribute shall not be the null value.

5) The *ST_PrivateInteriorShells* attribute shall not be the null value. The elements in the *ST_PrivateInteriorShells* attribute shall not be the null value. If the *ST_BRepSolid* value does not have interior shells, then the *ST_PrivateInteriorShells* attribute is set to an empty *ST_Surface* ARRAY value.

6) All the *ST_Surface* values in the *ST_PrivateExteriorShell* attribute and *ST_PrivateInteriorShells* attribute shall be in the same spatial reference system as the *ST_BRepSolid* value.

7) The coordinate dimension of an *ST_BRepSolid* value is equal to the coordinate dimension of its *ST_Surface* values.

8) An *ST_BRepSolid* value is simple.

9) The shell in the *ST_PrivateExteriorShell* attribute and the shells in the *ST_PrivateInteriorShells* attribute represent the boundary of the *ST_BRepSolid* value.

10) An *ST_BRepSolid* value is topologically closed.

11) The shells in the boundary may spatially intersect at most only a single point:

$$\forall\ p \in \mathit{ST\_BRepSolid},\ \forall\ c_1, c_2 \in \text{Boundary}(p),\ c_1 \neq c_2,$$
$$\forall\ a_1, a_2 \in \mathit{ST\_Point},\ a_1, a_2 \in c_1,\ a_1 \neq a_2,\ [\ a_1 \in c_2 \Rightarrow a_2 \notin c_2\ ]$$

12) An *ST_BRepSolid* value shall not have cut lines, spikes or punctures:

$$\forall\ p \in \mathit{ST\_BRepSolid},\ p = \text{Closure}(\text{Interior}(p))$$

13) The interior of every *ST_BRepSolid* value is a connected point set.

14) The exterior of an *ST_BRepSolid* with one or more holes is not connected. Each hole defines a disconnected component of the exterior.

15) An *ST_BRepSolid* is a topologically closed point set.

16) An *ST_BRepSolid* value returned by the constructor function corresponds to the empty set.

17) An *ST_BRepSolid* value corresponds to the empty set if the *ST_PrivateExteriorShell* attribute corresponds to the empty set.

18) An *ST_BRepSolid* value is well formed only if all the *ST_Surface* values in the *ST_PrivateExteriorShell* attribute and *ST_PrivateInteriorShells* attribute are well formed.

9.2.2 ST_BRepSolid Methods

Purpose

Return an ST_BRepSolid value constructed from either the well-known text representation, the well-known binary representation, a GML representation, or the specified ST_Surface values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN NEW ST_BRepSolid(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN NEW ST_BRepSolid(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN ST_BRepFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface)
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN SELF.ST_SRID(0).ST_ExteriorShell(asurface).
      ST_InteriorShells(CAST(ARRAY[] AS
         ST_Surface ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface,
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorShell(asurface).
      ST_InteriorShells(CAST(ARRAY[] AS
         ST_Surface ARRAY[ST_MaxGeometryArrayElements]))

CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface,
    asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN SELF.ST_SRID(0).ST_ExteriorShell(asurface).
      ST_InteriorShells(asurfacearray)
```

```
CREATE CONSTRUCTOR METHOD ST_BRepSolid
   (asurface ST_Surface,
    asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   RETURN SELF.ST_SRID(ansrid).ST_ExteriorShell(asurface).
      ST_InteriorShells(asurfacearray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_BRepSolid(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_BRepSolid(awktorgml, 0)*.
3) The method *ST_BRepSolid(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Solid XML elementin the GML representation, then return the result of the value expression: *ST_BRepFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_BRepFromText(awktorgml, ansrid)*.
5) The method *ST_BRepSolid(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_BRepSolid(awkb, 0)*.
7) The method *ST_BRepSolid(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
8) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_BRepFromWKB(awktorgml, ansrid)*.
9) The method *ST_BRepSolid(ST_Surface)* takes the following input parameters:
   b) an *ST_Surface* value *asurface*.
10) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(ST_Surface)* returns an *ST_BRepSolid* value with:
    a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorShell(ST_Surface)*:

i) the *ST_PrivateDimension* attribute set to 3.

ii) the *ST_PrivateCoordinateDimension* attribute set to 3.

iii) the *ST_PrivateIs3D* attribute set to 1 (one).

iv) the *ST_PrivateIsMeasured* attribute set to 0 (zero).

v) the *ST_PrivateExteriorShell* attribute set to *asurface.*

c) Using the method *ST_InteriorShells(ST_Surface ARRAY)*, the *ST_PrivateInteriorShells* attribute set to an empty *ST_Surface* ARRAY value.

11) The method *ST_BRepSolid(ST_Surface, INTEGER)* takes the following input parameters:

a) an *ST_Surface* value *asurface*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(ST_Surface, INTEGER)* returns an *ST_BRepSolid* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_ExteriorShell(ST_Surface)*:

i) the *ST_PrivateDimension* attribute set to 3.

ii) the *ST_PrivateCoordinateDimension* attribute set to 3.

iii) the *ST_PrivateIs3D* attribute set to 1 (one).

iv) the *ST_PrivateIsMeasured* attribute set to 0 (zero).

v) the *ST_PrivateExteriorShell* attribute set to *asurface.*

c) Using the method *ST_InteriorShells(ST_Surface ARRAY)*, the *ST_PrivateInteriorShells* attribute set to an empty *ST_Surface* ARRAY value.

13) The method *ST_BRepSolid(ST_Surface, ST_Surface ARRAY)* takes the following input parameters:

a) an *ST_Surface* value *asurface*,

b) an *ST_Surface* ARRAY value *asurfacearray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(ST_Surface, ST_Surface ARRAY)* returns an *ST_BRepSolid* value with:

a) The spatial reference system identifier set to 0 (zero).

b) Using the method *ST_ExteriorShell(ST_Surface)*:

i) the *ST_PrivateDimension* attribute set to 3.

ii) the *ST_PrivateCoordinateDimension* attribute set to 3.

iii) the *ST_PrivateIs3D* attribute set to 1 (one).

iv) the *ST_PrivateIsMeasured* attribute set to 0 (zero).

v) the *ST_PrivateExteriorShell* attribute set to *asurface.*

c) Using the method *ST_InteriorShells(ST_Surface ARRAY)*, the *ST_PrivateInteriorShells* attribute set to *asurfacearray*.

15) The method *ST_BRepSolid(ST_Surface, ST_Surface ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Surface* value *asurface*,

b) an *ST_Surface* ARRAY value *asurfacearray*,

c) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_BRepSolid(ST_Surface, ST_Surface ARRAY, INTEGER)* returns an *ST_BRepSolid* value with:

   a) The spatial reference system identifier set to *ansrid*.

   b) Using the method *ST_ExteriorShell(ST_Surface)*:

      i) the *ST_PrivateDimension* attribute set to 3.

      ii) the *ST_PrivateCoordinateDimension* attribute set to 3.

      iii) the *ST_PrivateIs3D* attribute set to 1 (one).

      iv) the *ST_PrivateIsMeasured* attribute set to 0 (zero).

      v) the *ST_PrivateExteriorShell* attribute set to *asurface.*

   c) Using the method *ST_InteriorShells(ST_Surface ARRAY)*, the *ST_PrivateInteriorShells* attribute set to *asurfacearray*.

9.2.3 ST_ExteriorShell Methods

Purpose

Observe and mutate the ST_PrivateExteriorShell attribute of an ST_BRepSolid value.

Definition

```
CREATE METHOD ST_ExteriorShell()
   RETURNS ST_Surface
   FOR ST_BRepSolid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateExteriorShell
      END

CREATE METHOD ST_ExteriorShell
   (asurface ST_Surface)
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   BEGIN
      DECLARE acounter INTEGER;

      IF asurface IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_BRepSolid);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and asurface.
      IF SELF.ST_SRID() <> asurface.ST_SRID() THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- If asurface is not a shell, then raise an exception.
      IF asurface.ST_IsShell() = 0 THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      -- For all interior shells
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(SELF.ST_InteriorShells()) DO
         -- If the current interior shell is not within
         -- asurface as a brep solid, then raise an exception
         IF SELF.ST_InteriorShells()[acounter].ST_Within(
            SELF.ST_BRepSolid(asurface, SELF.ST_SRID())) = 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         -- If the current interior shell intersects asurface
         -- with a dimension greater than 0 (zero), then
         -- raise an exception.
         IF SELF.ST_InteriorShells()[acounter].ST_Intersection(asurface).
            ST_Dimension() > 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
```

```
            END IF;
            SET acounter = acounter + 1;
         END WHILE;
         -- Return an ST_BRepSolid value with the ST_PrivateExteriorShell
         -- attribute set to asurface.
         RETURN
            SELF.ST_PrivateDimension(3).
               ST_PrivateCoordinateDimension(3).
               ST_PrivateIs3D(1).
               ST_PrivateIsMeasured(0).
               ST_PrivateExteriorShell(asurface);
      END
```

Description

1) The method *ST_ExteriorShell()* has no input parameters.

2) For the null-call method *ST_ExteriorShell()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateExteriorShell* attribute of SELF.

3) The method *ST_ExteriorShell(ST_Surface)* takes the following input parameters:

   a) an *ST_Surface* value *asurface*.

4) For the type-preserving method *ST_ExteriorShell(ST_Surface)*:

   Case:

   a) If *asurface* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) If the spatial reference system of SELF is not equal to the spatial reference system of *asurface*, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

   d) If *asurface* is not a shell, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   e) If any two shells in *asurface* and the interior shells of SELF spatially intersect with dimension of the result greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   f) If any shell in *asurfacearray* is not spatially within an *ST_BRepSolid* value formed from the exterior shell of SELF, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   g) Otherwise, return an *ST_BRepSolid* value with:

      i) The dimension set to 3.

      ii) The coordinate dimension set to 3.

      iii) The *ST_PrivateIs3D* attribute set to 1 (one).

      iv) The *ST_PrivateIsMeasured* attribute set to 0 (zero).

      v) The *ST_PrivateExteriorShell* attribute set to *asurface*.

9.2.4 ST_InteriorShells Methods

Purpose

Observe and mutate the ST_PrivateInteriorShells attribute of an ST_BRepSolid value.

Definition

```
CREATE METHOD ST_InteriorShells()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_BRepSolid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateInteriorShells
      END

CREATE METHOD ST_InteriorShells
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_BRepSolid
   FOR ST_BRepSolid
   BEGIN
      DECLARE acounter INTEGER;
      DECLARE bcounter INTEGER;

      IF SELF.ST_ExteriorShell() IS NULL THEN
         SIGNAL SQLSTATE '2FF98'
            SET MESSAGE_TEXT = 'null exterior shell';
      END IF;
      -- If asurfacearray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(asurfacearray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN SELF;
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and asurfacearray.
      IF (CARDINALITY(asurfacearray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(asurfacearray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      -- If any ST_Surface value is not a shell, then
      -- raise an exception.
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(asurfacearray) DO
         IF asurfacearray[acounter].ST_IsShell() = 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         SET acounter = acounter + 1;
      END WHILE;
      -- For all shells in asurfacearray
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(asurfacearray) DO
         -- If the current interior shell is not within
         -- the exterior shell as a brep solid, then raise an exception
         IF asurfacearray[acounter].ST_Within(
            SELF.ST_BRepSolid(SELF.ST_ExteriorShell(),
```

```
            SELF.ST_SRID())) = 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         -- If the current interior shell intersects the exterior
         -- shell with a dimension greater than zero, then
         -- raise an exception.
         IF asurfacearray[acounter].ST_Intersection(
            SELF.ST_ExteriorShell()).ST_Dimension() > 0 THEN
            SIGNAL SQLSTATE '2FF02'
               SET MESSAGE_TEXT = 'invalid argument';
         END IF;
         SET acounter = acounter + 1;
      END WHILE;
      SET acounter = 1;
      -- For each shell pair in asurfacearray
      WHILE acounter <= CARDINALITY(asurfacearray)-1 DO
         SET bcounter = acounter+1;
         WHILE bcounter <= CARDINALITY(asurfacearray) DO
            -- If the current interior shell pair overlap, then
            -- raise an exception.
            IF SELF.ST_BRepSolid(asurfacearray[acounter],
               SELF.ST_SRID()).ST_Overlaps(
               SELF.ST_BRepSolid(asurfacearray[bcounter],
               SELF.ST_SRID())) = 1 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            -- If the current interior shell pair intersect
            -- with a dimension greater than zero, then
            -- raise an exception.
            IF asurfacearray[acounter].ST_Intersection(
               asurfacearray[bcounter]).ST_Dimension() > 0 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            SET bcounter = bcounter + 1;
         END WHILE;
         SET acounter = acounter + 1;
      END WHILE;
      -- Return an ST_BRepSolid value with the ST_PrivateInteriorShells
      -- attribute set to asurfacearray.
      RETURN SELF.ST_PrivateInteriorShells(asurfacearray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_InteriorShells()* has no input parameters.

2) For the null-call method *ST_InteriorShells()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateInteriorShells* attribute of SELF.

3) The method *ST_InteriorShells(ST_Surface ARRAY)* takes the following input parameters:

   a) an *ST_Surface* ARRAY value *asurfacearray*.

4) For the type-preserving method *ST_InteriorShells(ST_Surface ARRAY)*:

   Case:

   a) If *SELF.ST_ExteriorShell()* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null exterior shell*.

   b) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *asurfacearray* is the null value or contains null elements.

   c) If SELF is the null value, then return the null value.

   d) If the cardinality of *asurfacearray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID*(*asurfacearray*), then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

   e) If any *ST_Surface* value in *asurfacearray* is not a shell, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   f) If any shells in *asurfacearray* and the exterior shell of SELF spatially intersect with dimension of the result greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   g) If any shell in *asurfacearray* is not spatially within an *ST_BRepSolid* value formed from the exterior shell of SELF, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   h) If any two shells in *asurfacearray*, formed into *ST_BRepSolid* values with no interior shells spatially overlap, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   i) If the intersection of any two shells in *asurfacearray* has a dimension greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   j) Otherwise, return an *ST_BRepSolid* value with the *ST_PrivateInteriorShells* attribute set to *asurfacearray*.

9.2.5 ST_NumIntShells Method

Purpose

Return the cardinality of the ST_PrivateInteriorShells attribute of an ST_BRepSolid value.

Definition

```
CREATE METHOD ST_NumIntShells()
   RETURNS INTEGER
   FOR ST_BRepSolid
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivateInteriorShells)
      END
```

Description

1) The method *ST_NumIntShells()* has no input parameters.

2) For the null-call method *ST_ NumIntShells()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the cardinality of the *ST_PrivateInteriorShells* attribute.

9.2.6 ST_InteriorShellN Method

Purpose

Return the specified element in the ST_PrivateInteriorShells attribute of an ST_BRepSolid value.

Definition

```
CREATE METHOD ST_InteriorShellN
   (aposition INTEGER)
   RETURNS ST_Surface
   FOR ST_BRepSolid
   BEGIN
      IF SELF.ST_IsEmpty() = 1  THEN
         RETURN CAST (NULL AS ST_Surface);
      END IF;
      IF aposition < 1 OR
         aposition > CARDINALITY(SELF.ST_PrivateInteriorShells) THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Surface);
         END;
      END IF;
      RETURN SELF.ST_PrivateInteriorShells[aposition];
   END
```

Description

1) The method *ST_InteriorShellN(INTEGER)* takes the following input parameters:
   a) an INTEGER value *aposition*.
2) For the null-call method *ST_InteriorShellN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If *aposition* is less than one or greater than the cardinality of the *ST_PrivateInteriorShells* attribute, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return an *ST_Surface* value at element *aposition* in the *ST_PrivateInteriorShells* attribute of SELF.

9.2.7 ST_BRepFromText Functions

Purpose

Return an ST_BRepSolid value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_BRepSolid value.

Definition

```
CREATE FUNCTION ST_BRepFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BRepFromText(awkt, 0)

CREATE FUNCTION ST_BRepFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_BRepFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_BRepFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_BRepFromText(awkt, 0)*.
3) The function *ST_BRepFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BRepFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_BRepSolid* value.

      If *awkt* is not producible in the BNF for <brepsolid text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_BRepSolid)*.

9.2.8 ST_BRepFromWKB Functions

Purpose

Return an ST_BRepSolid value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_BRepSolid value.

Definition

```
CREATE FUNCTION ST_BRepFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BRepFromWKB(awkb, 0)

CREATE FUNCTION ST_BRepFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_BRepFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_BRepFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_BRepFromWKB(awkb, 0)*.
3) The function *ST_BRepFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BRepFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_BRepSolid* value.

      If *awkb* is not producible in the BNF for <brepsolid binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_BRepSolid)*.

9.2.9 ST_BRepFromGML Functions

Purpose

Return an ST_BRepSolid value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Solid representation of an ST_BRepSolid value.

Definition

```
CREATE FUNCTION ST_BRepFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BRepFromGML(agml, 0)

CREATE FUNCTION ST_BRepFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_BRepSolid
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_BRepFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_BRepFromGML*(CHARACTER LARGE OBJECT) returns the result of the value expression: *ST_BRepFromGML(agml, 0)*.
3) The function *ST_BRepFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BRepFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a Solid XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_BRepSolid)*.

10 Geometry Collection Types

10.1 ST_GeomCollection Type and Routines

10.1.1 ST_GeomCollection Type

Purpose

The ST_GeomCollection type is a subtype of ST_Geometry and represents a collection of zero or more ST_Geometry values.

Definition

```
CREATE TYPE ST_GeomCollection
   UNDER ST_Geometry
   AS (
      ST_PrivateGeometries ST_Geometry
         ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_GeomCollection
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_GeomCollection
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_GeomCollection
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_GeomCollection
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_GeomCollection
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_GeomCollection
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_GeomCollection
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_GeomCollection
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_GeomCollection
   (ageometry ST_Geometry)
   RETURNS ST_GeomCollection
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_GeomCollection
   (ageometry ST_Geometry,
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_GeomCollection
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeomCollection
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_GeomCollection
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Geometries()
   RETURNS ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Geometries
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeomCollection
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_NumGeometries()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_GeometryN
   (aposition INTEGER)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.
4) The attribute *ST_PrivateGeometries* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateGeometries*.

Description

1) The *ST_GeomCollection* type provides for public use:
   a) a method *ST_GeomCollection(CHARACTER LARGE OBJECT)*,
   b) a method *ST_GeomCollection(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_GeomCollection(BINARY LARGE OBJECT)*,
   d) a method *ST_GeomCollection(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_GeomCollection(ST_Geometry)*,
   f) a method *ST_GeomCollection(ST_Geometry, INTEGER)*,
   g) a method *ST_GeomCollection(ST_Geometry ARRAY)*,
   h) a method *ST_GeomCollection(ST_Geometry ARRAY, INTEGER)*,
   i) a method *ST_Geometries()*,
   j) a method *ST_Geometries(ST_Geometry ARRAY)*,
   k) a method *ST_NumGeometries()*,
   l) a method *ST_GeometryN(INTEGER)*,
   m) a function *ST_GeomCollFromTxt(CHARACTER LARGE OBJECT)*,
   n) a function *ST_GeomCollFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,
   o) a function *ST_GeomCollFromWKB(BINARY LARGE OBJECT)*,
   p) a function *ST_GeomCollFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   q) a function *ST_GeomCollFromGML(CHARACTER LARGE OBJECT)*,
   r) a function *ST_GeomCollFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The *ST_PrivateGeometries* attribute contains the collection of *ST_Geometry* values.
3) The *ST_PrivateGeometries* attribute shall not be the null value. The elements in the *ST_PrivateGeometries* attribute shall not be the null value.

4) The coordinate dimension of an *ST_GeomCollection* value is equal to the coordinate dimension of its *ST_Geometry* values.

5) The dimension of an *ST_GeomCollection* value is the maximum dimension value of all the *ST_Geometry* values in the *ST_PrivateGeometries* attribute.

6) An *ST_GeomCollection* value returned by the constructor function corresponds to the empty set.

7) An *ST_GeomCollection* value with no elements in the *ST_PrivateGeometries* attribute corresponds to the empty set.

8) Subtypes of *ST_GeomCollection* may restrict membership based on dimension and may place other constraints such as the degree that the elements spatially intersect between *ST_Geometry* values.

9) A value with the most specific type of *ST_GeomCollection* is simple if:

   a) all the elements in the *ST_PrivateGeometries* attribute are simple.

   b) the interior of any element in the *ST_PrivateGeometries* attribute does not intersect the interior of any other element in the *ST_PrivateGeometries* attribute.

10) An *ST_GeomCollection* value is well formed only if all of the *ST_Geometry* values in *ST_PrivateGeometries* attribute are well formed.

11) All the *ST_Geometry* values in the *ST_PrivateGeometries* attribute shall be in the same spatial reference system as the *ST_GeomCollection* value.

10.1.2 ST_GeomCollection Methods

Purpose

Return an ST_GeomCollection value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Geometry values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN NEW ST_GeomCollection(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN NEW ST_GeomCollection(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN ST_GeomCollFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (ageometry ST_Geometry)
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN SELF.ST_SRID(ageometry.ST_SRID()).
      ST_Geometries(ARRAY[ageometry])

CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (ageometry ST_Geometry,
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(ARRAY[ageometry])

CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN SELF.ST_SRID(ST_CheckSRID(ageometryarray)).
      ST_Geometries(ageometryarray)
```

```
CREATE CONSTRUCTOR METHOD ST_GeomCollection
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(ageometryarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_GeomCollection(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_GeomCollection(awktorgml, 0)*.
3) The method *ST_GeomCollection(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awktorgml*,
   b) an INTEGER value *ansrid*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a MultiGeometry XML element in the GML representation, then return the result of the value expression: *ST_GeomCollFromGML(awktorgml, ansrid)*.
   b) Otherwise, return the result of the value expression: *ST_GeomCollFromTxt(awktorgml, ansrid)*.
5) The method *ST_GeomCollection(BINARY LARGE OBJECT)* takes the following input parameter:
   a) a BINARY LARGE OBJECT value *awkb*.
6) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_GeomCollection(awkb, 0)*.
7) The method *ST_GeomCollection(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
8) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_GeomCollFromWKB(awkb, ansrid)*.
9) The method *ST_GeomCollection(ST_Geometry)* takes the following input parameters:
   a) an *ST_Geometry* value *ageometry*.
10) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(ST_Geometry)* returns the result of the value expression: *NEW ST_GeomCollection(acurve, 0).*
11) The method *ST_GeomCollection(ST_Geometry, INTEGER)* takes the following input parameters:

a) an *ST_Geometry* value *ageometry*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(ST_Geometry, INTEGER)* returns the result of the value expression: *NEW ST_GeomCollection(ARRAY[ageometry], ansrid).*

13) The method *ST_GeomCollection(ST_Geometry ARRAY)* takes the following input parameters:

a) an *ST_Geometry* ARRAY value *ageometryarray*.

14) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(ST_Geometry ARRAY)* returns the result of the value expression: *NEW ST_GeomCollection(ageometryarray, 0).*

15) The method *ST_GeomCollection(ST_Geometry ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_Geometry* ARRAY value *ageometryarray*,

b) an INTEGER value *ansrid*.

16) The null-call type-preserving SQL-invoked constructor method *ST_GeomCollection(ST_Geometry ARRAY, INTEGER)* returns an *ST_GeomCollection* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Geometries(ST_Geometry ARRAY)*:

i) the *ST_PrivateDimension* attribute set to *ST_MaxDimension(ageometryarray)*.

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(ageometryarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(ageometryarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(ageometryarray)*.

v) the *ST_PrivateGeometries* attribute set to *ageometryarray*.

10.1.3 ST_Geometries Methods

Purpose

Observe and mutate the ST_PrivateGeometries attribute of an ST_GeomCollection value.

Definition

```
CREATE METHOD ST_Geometries()
   RETURNS ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_GeomCollection
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateGeometries
      END

CREATE METHOD ST_Geometries
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeomCollection
   FOR ST_GeomCollection
   BEGIN
      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_GeomCollection);
      END IF;
      -- Check that there are no mixed spatial reference
      -- systems between SELF and ageometryarray.
      IF (CARDINALITY(ageometryarray) > 0) AND
         (SELF.ST_SRID() <> ST_CheckSRID(ageometryarray)) THEN
         SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      RETURN
         SELF.ST_PrivateDimension(ST_MaxDimension(ageometryarray)).
            ST_PrivateCoordinateDimension(ST_GetCoordDim(ageometryarray)).
            ST_PrivateIs3D(ST_GetIs3D(ageometryarray)).
            ST_PrivateIsMeasured(ST_GetIsMeasured(ageometryarray)).
            ST_PrivateGeometries(ageometryarray);
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Geometries()* has no input parameters.
2) For the null-call method *ST_Geometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the value of the *ST_PrivateGeometries* attribute.
3) The method *ST_Geometries(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.

4) For the type-preserving method *ST_Geometries(ST_Geometry ARRAY)*:
   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.
   b) Case:
      i) If SELF is the null value, then return the null value.
      ii) If the cardinality of *ageometryarray* is greater than 0 (zero) and the spatial reference system of SELF is not equal to *ST_CheckSRID(ageometryarray)*, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.
      iii) Otherwise, return an *ST_GeomCollection* value with:
         1) The dimension set to *ST_MaxDimension(ageometryarray)*.
         2) The coordinate dimension set to the value expression: *ST_GetCoordDim(ageometryarray)*.
         3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(ageometryarray)*.
         4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(ageometryarray)*.
         5) The *ST_PrivateGeometries* attribute set to *ageometryarray*.

10.1.4 ST_NumGeometries Method

Purpose

Return the cardinality of the ST_PrivateGeometries attribute of an ST_GeomCollection value.

Definition

```
CREATE METHOD ST_NumGeometries()
   RETURNS INTEGER
   FOR ST_GeomCollection
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_PrivateGeometries)
      END
```

Description

1) The method *ST_NumGeometries()* has no input parameters.
2) For the null-call method *ST_NumGeometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the cardinality of the *ST_PrivateGeometries* attribute.

10.1.5 ST_GeometryN Method

Purpose

Return the specified ST_Geometry value in the ST_PrivateGeometries attribute of an ST_GeomCollection value.

Definition

```
CREATE METHOD ST_GeometryN
   (aposition INTEGER)
   RETURNS ST_Geometry
   FOR ST_GeomCollection
   BEGIN
      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS ST_Geometry);
      END IF;
      IF aposition < 1 OR
         aposition > CARDINALITY(SELF.ST_PrivateGeometries) THEN
         BEGIN
            SIGNAL SQLSTATE '01F01'
               SET MESSAGE_TEXT = 'invalid position';
            RETURN CAST (NULL AS ST_Geometry);
         END;
      END IF;
      RETURN SELF.ST_PrivateGeometries[aposition];
   END
```

Description

1) The method *ST_GeometryN(INTEGER)* takes the following input parameters:
   a) an INTEGER value a*position*.
2) For the null-call method *ST_GeometryN(INTEGER)*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) If aposition is less than one or greater than the cardinality of the *ST_PrivateGeometries* attribute, then:
      i) A completion condition is raised: *SQL/MM Spatial warning – invalid position*.
      ii) Return the null value.
   c) Otherwise, return the element of the *ST_PrivateGeometries* attribute at position aposition.

10.1.6 ST_GeomCollFromTxt Functions

Purpose

Return an ST_GeomCollection value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_GeomCollection value.

Definition

```
CREATE FUNCTION ST_GeomCollFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeomCollFromText(awkt, 0)

CREATE FUNCTION ST_GeomCollFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_GeomCollFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_GeomCollFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_GeomCollFromTxt(awkt, 0)*.
3) The function *ST_GeomCollFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeomCollFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_GeomCollection* value.

      If *awkt* is not producible in the BNF for <geometrycollection text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_GeomCollection)*.

10.1.7 ST_GeomCollFromWKB Functions

Purpose

Return an ST_GeomCollection value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_GeomCollection value.

Definition

```
CREATE FUNCTION ST_GeomCollFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeomCollFromWKB(awkb, 0)

CREATE FUNCTION ST_GeomCollFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_GeomCollFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_GeomCollFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_GeomCollFromWKB(awkb, 0)*.
3) The function *ST_GeomCollFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeomCollFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_GeomCollection* value.

      If *awkb* is not producible in the BNF for <geometrycollection binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_GeomCollection)*.

10.1.8 ST_GeomCollFromGML Functions

Purpose

Return an ST_GeomCollection value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representationof an ST_GeomCollection value.

Definition

```
CREATE FUNCTION ST_GeomCollFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_GeomCollFromGML(agml, 0)

CREATE FUNCTION ST_GeomCollFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_GeomCollection
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_GeomCollFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_GeomCollFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_GeomCollFromGML(agml, 0)*.
3) The function *ST_GeomCollFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_GeomCollFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a MultiGeometry XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_GeomCollection)*.

10.2 ST_MultiPoint Type and Routines

10.2.1 ST_MultiPoint Type

Purpose

The ST_MultiPoint type is a 0-dimensional geometry and represents a collection of ST_Point values.

Definition

```
CREATE TYPE ST_MultiPoint
   UNDER ST_GeomCollection
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_MultiPoint
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_MultiPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPoint
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_MultiPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPoint
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_MultiPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPoint
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_MultiPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPoint
      (apointarray ST_Point
         ARRAY[ST_MaxGeometryArrayElements])
      RETURNS ST_MultiPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```



```
CONSTRUCTOR METHOD ST_MultiPoint
   (apointarray ST_Point
      ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_Geometries()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_Geometries
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPoint
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The *ST_MultiPoint* type provides for public use:
   a) a method *ST_MultiPoint(CHARACTER LARGE OBJECT)*,
   b) a method *ST_MultiPoint(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_MultiPoint(BINARY LARGE OBJECT)*,
   d) a method *ST_MultiPoint(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_MultiPoint(ST_Point ARRAY)*,
   f) a method *ST_MultiPoint(ST_Point ARRAY, INTEGER)*,
   g) an overriding method *ST_Geometries()*,
   h) an overriding method *ST_Geometries(ST_Geometry ARRAY)*,
   i) a function *ST_MPointFromText(CHARACTER LARGE OBJECT)*,
   j) a function *ST_MPointFromText(CHARACTER LARGE OBJECT, INTEGER)*,
   k) a function *ST_MPointFromWKB(BINARY LARGE OBJECT)*,
   l) a function *ST_MPointFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   m) a function *ST_MPointFromGML(CHARACTER LARGE OBJECT)*,
   n) a function *ST_MPointFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The dimension of an *ST_MultiPoint* value is 0 (zero).
3) The elements of the *ST_PrivateGeometries* attribute are restricted to *ST_Point* values.
4) The *ST_Point* values in the *ST_PrivateGeometries* attribute are not connected or ordered.
5) If no two *ST_Point* values in the *ST_MultiPoint* value are equal, then the *ST_MultiPoint* value is simple.

a) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
b) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

6) The boundary of an *ST_MultiPoint* value is the empty set.
7) An *ST_MultiPoint* value is well formed only if and only if all of the *ST_Point* values in the *ST_PrivateGeometries* attribute are well formed.
8) An *ST_MultiPoint* value returned by the constructor function corresponds to the empty set.

10.2.2 ST_MultiPoint Methods

Purpose

Return an ST_MultiPoint value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Point values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_MultiPoint
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   RETURN NEW ST_MultiPoint(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_MultiPoint
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_MultiPoint
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   RETURN NEW ST_MultiPoint(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_MultiPoint
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   RETURN ST_MPointFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_MultiPoint
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   RETURN SELF.ST_SRID(0).ST_Geometries(apointarray)

CREATE CONSTRUCTOR METHOD ST_MultiPoint
   (apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(apointarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_MultiPoint(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_MultiPoint(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiPoint(awktorgml, 0)*.

3) The method *ST_MultiPoint(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_MultiPoint(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a MultiPoint XML element in the GML representation, then return the result of the value expression: *ST_MPointFromGML*(*awktorgml*, *ansrid*).

   b) Otherwise, return the result of the value expression: *ST_MPointFromText*(*awktorgml*, *ansrid*).

5) The method *ST_MultiPoint(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_MultiPoint(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiPoint(awktorgml, 0)*.

7) The method *ST_MultiPoint(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_MultiPoint(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_MPointFromWKB(awkb, ansrid)*.

9) The method *ST_MultiPoint(ST_Point ARRAY)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_MultiPoint(ST_Point ARRAY)* returns the result of the value expression: *NEW ST_MultiPoint(apointarray, 0).*

11) The method *ST_MultiPoint(ST_Point ARRAY, INTEGER)* takes the following input parameters:

   a) an *ST_Point* ARRAY value *apointarray*,

   b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_MultiPoint(ST_Point ARRAY, INTEGER)* returns an *ST_MultiPoint* value with:

   a) The spatial reference system identifier set to *ansrid*.

   b) Using the method *ST_Geometries(ST_Geometry ARRAY)*:

      i) the *ST_PrivateDimension* attribute set to 0 (zero).

      ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apointarray).*

      iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apointarray)*.

      iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apointarray)*.

      v) the *ST_PrivateGeometries* attribute set to *apointarray*.

10.2.3 ST_Geometries Methods

Purpose

Observe and mutate the ST_PrivateGeometries attribute of an ST_MultiPoint value.

Definition

```
CREATE METHOD ST_Geometries()
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_MultiPoint
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateGeometries AS
               ST_Point ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_Geometries
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPoint
   FOR ST_MultiPoint
   BEGIN
      DECLARE apointarray ST_Point
         ARRAY[ST_MaxGeometryArrayElements];

      -- Cast ageometryarray to an ST_Point ARRAY
      SET apointarray = CAST(ageometryarray AS
         ST_Point ARRAY[ST_MaxGeometryArrayElements]);
      -- If SELF is the null value, then return the null value.
      -- Otherwise, return an ST_MultiPoint value containing
      -- apointarray.
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               (SELF AS ST_GeomCollection).
                  ST_Geometries(apointarray)
         END;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Geometries()* has no input parameters.

2) For the null-call method *ST_Geometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the value of the *ST_PrivateGeometries* attribute as an *ST_Point* ARRAY.

3) The method *ST_Geometries(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry ARRAY* value *ageometryarray*.

4) For the type-preserving method *ST_Geometries(ST_Geometry ARRAY)*:

a) Let *APOINTARRAY* be the result of casting *ageometryarray* to an *ST_Point* ARRAY value (implicitly using *ST_ToPointAry(ST_Geometry ARRAY)*).

b) Case:

   i) If SELF is the null value, then return the null value.

   ii) Otherwise, using the method *ST_Geometries(ST_Geometry ARRAY)* for type *ST_GeomCollection*, return an *ST_MultiPoint* value with:

      1) The dimension set to 0 (zero).

      2) The coordinate dimension set to the value expression: *ST_GetCoordDim(APOINTARRAY)*.

      3) The *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(APOINTARRAY)*.

      4) The *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(APOINTARRAY)*.

      5) The *ST_PrivateGeometries* attribute set to *APOINTARRAY*.

10.2.4 ST_MPointFromText Functions

Purpose

Return an ST_MultiPoint value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiPoint value.

Definition

```
CREATE FUNCTION ST_MPointFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MPointFromText(awkt, 0)

CREATE FUNCTION ST_MPointFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_MPointFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_MPointFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MPointFromText(awkt, 0)*.
3) The function *ST_MPointFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MPointFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_MultiPoint* value.

      If *awkt* is not producible in the BNF for <multipoint text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_MultiPoint)*.

10.2.5 ST_MPointFromWKB Functions

Purpose

Return an ST_MultiPoint value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiPoint value.

Definition

```
CREATE FUNCTION ST_MPointFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MPointFromWKB(awkb, 0)

CREATE FUNCTION ST_MPointFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_MPointFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_MPointFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_MPointFromWKB(awkb, 0)*.
3) The function *ST_MPointFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MPointFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiPoint* value.

      If *awkb* is not producible in the BNF for <multipoint binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_MultiPoint)*.

10.2.6 ST_MPointFromGML Functions

Purpose

Return an ST_MultiPoint value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representationof an ST_MultiPoint value.

Definition

```
CREATE FUNCTION ST_MPointFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MPointFromGML(agml, 0)

CREATE FUNCTION ST_MPointFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_MultiPoint
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_MPointFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_MPointFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MPointFromGML(agml, 0)*.
3) The function *ST_MPointFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MPointFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a MultiPoint XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_MultiPoint)*.

10.3 ST_MultiCurve Type and Routines

10.3.1 ST_MultiCurve Type

Purpose

The ST_MultiCurve type is a 1-dimensional geometry and represents a collection of ST_Curve.

Definition

```
CREATE TYPE ST_MultiCurve
   UNDER ST_GeomCollection
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_MultiCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_MultiCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_MultiCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_MultiCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiCurve
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_MultiCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiCurve
      (acurvearray ST_Curve
         ARRAY[ST_MaxGeometryArrayElements])
      RETURNS ST_MultiCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_MultiCurve
   (acurvearray ST_Curve
      ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsClosed()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DIsClosed()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Length()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Length
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DLength()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_PerpPoints
   (apoint ST_Point)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_Geometries()
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_Geometries
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiCurve
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The *ST_MultiCurve* type provides for public use:
   a) a method *ST_MultiCurve(CHARACTER LARGE OBJECT)*,
   b) a method *ST_MultiCurve(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_MultiCurve(BINARY LARGE OBJECT)*,
   d) a method *ST_MultiCurve(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_MultiCurve(ST_Curve ARRAY)*,
   f) a method *ST_MultiCurve(ST_Curve ARRAY, INTEGER)*,
   g) a method *ST_IsClosed()*,
   h) a method *ST_3DIsClosed()*,
   i) a method *ST_Length()*,
   j) a method *ST_Length(CHARACTER VARYING)*,
   k) a method *ST_3DLength()*,
   l) a method *ST_3DLength(CHARACTER VARYING)*,
   m) a method *ST_PerpPoints(ST_Point)*,
   n) an overriding method *ST_Geometries()*,
   o) an overriding method *ST_Geometries(ST_Geometry ARRAY)*,
   p) a function *ST_MCurveFromText(CHARACTER LARGE OBJECT)*,
   q) a function *ST_MCurveFromText(CHARACTER LARGE OBJECT, INTEGER)*,
   r) a function *ST_MCurveFromWKB(BINARY LARGE OBJECT)*,
   s) a function *ST_MCurveFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   t) a function *ST_MCurveFromGML(CHARACTER LARGE OBJECT)*,
   u) a function *ST_MCurveFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The dimension of an *ST_MultiCurve* value is 1 (one).

3) The elements of an *ST_MultiCurve* value are *ST_Curve* values.

4) If all of the elements in the *ST_PrivateGeometries* attribute are simple and any two elements only spatially intersect at the boundaries of both elements, then an *ST_MultiCurve* is simple.

   a) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation of *ST_IsSimple*.

   b) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation of *ST_3DIsSimple*.

   c) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation of *ST_3DIsSimple*.

5) The boundary of an *ST_MultiCurve* value is obtained by applying the mod 2 union rule: an *ST_Point* value is in the boundary of an *ST_MultiCurve* if it is in the boundaries of an odd number of elements of the *ST_MultiCurve*.

6) An *ST_MultiCurve* value is closed if all of its elements are closed. The boundary of a closed *ST_MultiCurve* is the empty set.

   a) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation of *ST_IsClosed*.

   b) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation of *ST_3DIsClosed*.

   c) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

7) An *ST_MultiCurve* value is defined as topologically closed.

8) An *ST_MultiCurve* value is well formed only if all of the *ST_Curve* values in the *ST_PrivateGeometries* attribute are well formed.

9) An *ST_MultiCurve* value returned by the constructor function corresponds to the empty set.

10.3.2 ST_MultiCurve Methods

Return an ST_MultiCurve value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Curve values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_MultiCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   RETURN NEW ST_MultiCurve(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_MultiCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_MultiCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   RETURN NEW ST_MultiCurve(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_MultiCurve
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   RETURN ST_MCurveFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_MultiCurve
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   RETURN SELF.ST_SRID(0).ST_Geometries(acurvearray)

CREATE CONSTRUCTOR METHOD ST_MultiCurve
   (acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(acurvearray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_MultiCurve(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_MultiCurve(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiCurve(awktorgml, 0)*.

3) The method *ST_MultiCurve(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_MultiCurve(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a MultiCurve XML element in the GML representation, then return the result of the value expression: *ST_MCurveFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_MCurveFromText(awktorgml, ansrid)*.

5) The method *ST_MultiCurve(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_MultiCurve(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiCurve(awkb, 0)*.

7) The method *ST_MultiCurve(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_MultiCurve(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_MCurveFromWKB(awkb, ansrid)*.

9) The method *ST_MultiCurve(ST_Curve ARRAY)* takes the following input parameters:

   a) an *ST_Curve* ARRAY value *acurvearray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_MultiCurve(ST_Curve ARRAY)* returns the result of the value expression: *NEW ST_MultiCurve(acurvearray, 0).*

11) The method *ST_MultiCurve(ST_Curve ARRAY, INTEGER)* takes the following input parameters:

    a) an *ST_Curve* ARRAY value *acurvearray*,

    b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_MultiCurve(ST_Curve ARRAY, INTEGER)* returns an *ST_MultiCurve* value with:

    a) The spatial reference system identification set to *ansrid*.

    b) Using the method *ST_Geometries(ST_Geometry ARRAY)*:

       i) the *ST_PrivateDimension* attribute set to 1 (one).

       ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(acurvearray).*

       iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(acurvearray)*.

       iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(acurvearray)*.

       v) the *ST_PrivateGeometries* attribute set to *acurvearray*.

10.3.3 ST_IsClosed Method

Purpose

Test if an ST_MultiCurve value is closed, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_IsClosed()
   RETURNS INTEGER
   FOR ST_MultiCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty = 1 THEN
            0
         ELSE
            SELF.ST_Boundary().ST_IsEmpty()
      END
```

Description

1) The method *ST_IsClosed()* has no input parameters.
2) The null-call method *ST_IsClosed()* returns:

   Case:

   a) If SELF is the empty set, then 0 (zero).
   b) If the boundary of the *ST_MultiCurve* value is the empty set, then 1 (one).
   c) Otherwise, 0 (zero).
3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.
4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

10.3.4 ST_3DIsClosed Method

Purpose

Test if an ST_MultiCurve value is closed, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DIsClosed()
   RETURNS INTEGER
   FOR ST_MultiCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty = 1 THEN
            0
         ELSE
            SELF.ST_3DBoundary().ST_IsEmpty()
      END
```

Description

1) The method *ST_3DIsClosed()* has no input parameters.

2) The null-call method *ST_3DIsClosed()* returns:

   Case:

   a) If SELF is the empty set, then 0 (zero).

   b) If the boundary of the *ST_MultiCurve* value is the empty set, then 1 (one).

   c) Otherwise, 0 (zero).

3) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.

4) If *SELF.ST_IsMeasured()* is equal to 1 (one), then the m coordinate values are not considered in the calculation.

10.3.5 ST_Length Methods

Purpose

Return the length measurement of an ST_MultiCurve value, ignoring z and m coordinate valuess in the calculations.

Definition

```
CREATE METHOD ST_Length()
   RETURNS DOUBLE PRECISION
   FOR ST_MultiCurve
   BEGIN
      DECLARE length DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET length = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET length = length + SELF.ST_GeometryN(counter).ST_Length();
         SET counter = counter + 1;
      END WHILE;
      RETURN length;
   END

CREATE METHOD ST_Length
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_MultiCurve
   BEGIN
      DECLARE length DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET length = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET length = length +
         SELF.ST_GeometryN(counter).ST_Length(aunit);
         SET counter = counter + 1;
      END WHILE;
      RETURN length;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Length()* has no input parameters.
2) For the null-call method *ST_Length()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the sum of the *ST_Length()* values of each element in the *ST_PrivateGeometries* attribute of SELF.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Length()* is in the linear unit of measure identified by <linear unit>.

ii) Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.

3) The method *ST_Length(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_Length(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the sum of the *ST_Length(aunit)* values of each element in *ST_PrivateGeometries* attribute of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the sum of the *ST_Length(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF.

e) Te returned value is in the units indicated by *aunit*.

5) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

10.3.6 ST_3DLength Methods

Purpose

Return the length measurement of an ST_MultiCurve value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DLength()
   RETURNS DOUBLE PRECISION
   FOR ST_MultiCurve
   BEGIN
      DECLARE length DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET length = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET length = length + SELF.ST_GeometryN(counter).ST_3DLength();
         SET counter = counter + 1;
      END WHILE;
      RETURN length;
   END

CREATE METHOD ST_3DLength
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_MultiCurve
   BEGIN
      DECLARE length DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET length = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET length = length +
         SELF.ST_GeometryN(counter).ST_3DLength(aunit);
         SET counter = counter + 1;
      END WHILE;
      RETURN length;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DLength()* has no input parameters.

2) For the null-call method *ST_3DLength()*:

   a) Case:

      i) If SELF is an empty set, then return the null value.

      ii) Otherwise, return the sum of the *ST_3DLength()* values of each element in the *ST_PrivateGeometries* attribute of SELF.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DLength()* is in the linear unit of measure identified by <linear unit>.

ii) Otherwise, the value returned by *ST_3DLength()* is in an implementation-defined unit of measure.

3) The method *ST_3DLength(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_3DLength(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute the sum of the *ST_3DLength(aunit)* values of each element in *ST_PrivateGeometries* attribute of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the sum of the *ST_3DLength(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF.

e) The returned value is in the units indicated by *aunit*.

5) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.

10.3.7 ST_PerpPoints Method

Purpose

Return the geometry representing the perpendicular projection of the given point on the multicurve, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_PerpPoints
   (apoint ST_Point)
   RETURNS ST_Geometry
   FOR ST_MultiCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_PerpPoints(ST_Point)* takes the following input parameter:
   a) an *ST_Point* value *apoint*.
2) For the null-call method *ST_PerpPoints(ST_Point)*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) If *apoint* is an empty set, then return the null value.
      iii) If SELF and *apoint* spatially intersect such that z and m coordinate values are not considered in the calculation, then return *apoint*.
      iv) If *apoint* cannot be perpendicularly projected on SELF, then return an empty set.
      v) Otherwise, return a geometry value representing the perpendicular projection of *apoint* on SELF, calculated in the spatial reference system of SELF, using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation or in the return values.

      NOTE The result of the projection algorithm may produce the following
      - an ST_Point value when it produces a single point result
      - an ST_MultiPoint value when it produces a finite number of points
      - an ST_Curve value when it produces a connected set of points
      - an ST_MultiCurve value when it produces a number of connected set of points
      - an ST_GeomCollection when it produces a mixture of point values and curve values.

10.3.8 ST_Geometries Methods

Purpose

Observe and mutate the ST_PrivateGeometries attribute of an ST_MultiCurve value.

Definition

```
CREATE METHOD ST_Geometries()
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_MultiCurve
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateGeometries AS ST_Curve
               ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_Geometries
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiCurve
   FOR ST_MultiCurve
   BEGIN
      DECLARE acurvearray ST_Curve
         ARRAY[ST_MaxGeometryArrayElements];

      -- Cast ageometryarray to an ST_Curve ARRAY
      SET acurvearray = CAST(ageometryarray AS
         ST_Curve ARRAY[ST_MaxGeometryArrayElements]);
      -- If SELF is the null value, then return the null value. Otherwise,
      -- return an ST_MultiCurve value containing acurvearray.
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               (SELF AS ST_GeomCollection).
                  ST_Geometries(acurvearray)
         END;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Geometries()* has no input parameters.
2) For the null-call method *ST_Geometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the value of the *ST_PrivateGeometries* attribute as an *ST_Curve ARRAY*.
3) The method *ST_Geometries(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry ARRAY* value *ageometryarray*.
4) For the type-preserving method *ST_Geometries(ST_Geometry ARRAY)*:
   a) Let *ACURVEARRAY* be the result of casting *ageometryarray* to an *ST_Curve* ARRAY value (implicitly using *ST_ToCurveAry(ST_Geometry ARRAY)*).

b) Case:

   i) If SELF is the null value, then return the null value.

   ii) Otherwise, return an *ST_MultiCurve* value with:

      1) The dimension set to 1 (one).

      2) Using the method *ST_Geometries(ST_Geometry ARRAY)* for type *ST_GeomCollection*, the *ST_PrivateGeometries* attribute set to *ACURVEARRAY*.

10.3.9 ST_MCurveFromText Functions

Purpose

Return an ST_MultiCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiCurve value.

Definition

```
CREATE FUNCTION ST_MCurveFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MCurveFromText(awkt, 0)

CREATE FUNCTION ST_MCurveFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_MCurveFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_MCurveFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MCurveFromText(awkt, 0)*.
3) The function *ST_MCurveFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MCurveFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_MultiCurve* value.

      If *awkt* is not producible in the BNF for <multicurve text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_MultiCurve)*.

10.3.10 ST_MCurveFromWKB Functions

Purpose

Return an ST_MultiCurve value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiCurve value.

Definition

```
CREATE FUNCTION ST_MCurveFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MCurveFromWKB(awkb, 0)

CREATE FUNCTION ST_MCurveFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_MCurveFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_MCurveFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_MCurveFromWKB(awkb, 0)*.
3) The function *ST_MCurveFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MCurveFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiCurve* value.

      If *awkb* is not producible in the BNF for <multicurve binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_MultiCurve)*.

10.3.11 ST_MCurveFromGML Functions

Purpose

Return an ST_MultiCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representationof an ST_MultiCurve value.

Definition

```
CREATE FUNCTION ST_MCurveFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MCurveFromGML(agml, 0)

CREATE FUNCTION ST_MCurveFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_MultiCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_MCurveFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_MCurveFromGML*(CHARACTER LARGE OBJECT) returns the result of the value expression: *ST_MCurveFromGML(agml, 0)*.
3) The function *ST_MCurveFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MCurveFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a MultiCurve XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_MultiCurve)*.

10.4 ST_MultiLineString Type and Routines

10.4.1 ST_MultiLineString Type

Purpose

The ST_MultiLineString type is a subtype of the ST_MultiCurve and represents a collection of ST_LineString values.

Definition

```
CREATE TYPE ST_MultiLineString
   UNDER ST_MultiCurve
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_MultiLineString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_MultiLineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiLineString
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_MultiLineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiLineString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_MultiLineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiLineString
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_MultiLineString
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_MultiLineString
   (alinestringarray ST_LineString
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiLineString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_MultiLineString
   (alinestringarray ST_LineString
      ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_Geometries()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_Geometries
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiLineString
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The *ST_MultiLineString* type provides for public use:
   a) a method *ST_MultiLineString(CHARACTER LARGE OBJECT)*,
   b) a method *ST_MultiLineString(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_MultiLineString(BINARY LARGE OBJECT)*,
   d) a method *ST_MultiLineString(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_MultiLineString(ST_LineString ARRAY)*,
   f) a method *ST_MultiLineString(ST_LineString ARRAY, INTEGER)*,
   g) an overriding method *ST_Geometries()*,
   h) an overriding method *ST_Geometries(ST_Geometry ARRAY)*,
   i) a function *ST_MLineFromText(CHARACTER LARGE OBJECT)*,
   j) a function *ST_MLineFromText(CHARACTER LARGE OBJECT, INTEGER)*,
   k) a function *ST_MLineFromWKB(BINARY LARGE OBJECT)*,
   l) a function *ST_MLineFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   m) a function *ST_MLineFromGML(CHARACTER LARGE OBJECT),*

n) a function *ST_MLineFromGML(CHARACTER LARGE OBJECT, INTEGER)*.

2) The elements of the *ST_PrivateGeometries* attribute are restricted to *ST_LineString* values.

3) An *ST_MultiLineString* value is well formed only if and only if all of the *ST_LineString* values in the *ST_PrivateGeometries* attribute are well formed.

4) An *ST_MultiLineString* value returned by the constructor function corresponds to the empty set.

10.4.2 ST_MultiLineString Methods

Purpose

Return an ST_MultiLineString value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_LineString values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_MultiLineString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   RETURN NEW ST_MultiLineString(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_MultiLineString
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_MultiLineString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   RETURN NEW ST_MultiLineString(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_MultiLineString
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   RETURN ST_MLineFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_MultiLineString
   (alinestringarray ST_LineString ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   RETURN SELF.ST_SRID(0).ST_Geometries(alinestringarray)

CREATE CONSTRUCTOR METHOD ST_MultiLineString
   (alinestringarray ST_LineString ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(alinestringarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_MultiLineString(CHARACTER LARGE OBJECT)* takes the following input parameter:

a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_MultiLineString(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiLineString(awktorgml, 0)*.

3) The method *ST_MultiLineString(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a CHARACTER LARGE OBJECT value *awktorgml*,

b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_MultiLineString(CHARACTER LARGE OBJECT, INTEGER)*:

Case:

a) If *awktorgml* contains a MultiLineString XML element in the GML representation, then return the result of the value expression: *ST_MLineStringFromGML(awktorgml, ansrid)*.

b) Otherwise, return the result of the value expression: *ST_MLineFromText(awktorgml, ansrid)*.

5) The method *ST_MultiLineString(BINARY LARGE OBJECT)* takes the following input parameter:

a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_MultiLineString(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiLineString(awkb, 0)*.

7) The method *ST_MultiLineString(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

a) a BINARY LARGE OBJECT value *awkb*,

b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_MultiLineString(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_MLineFromWKB(awkb, ansrid)*.

9) The method *ST_MultiLineString(ST_LineString ARRAY)* takes the following input parameters:

a) an *ST_LineString* ARRAY value *alinestringarray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_MultiLineString(ST_LineString ARRAY)* returns the result of the value expression: *NEW ST_MultiLineString(alinestringarray, 0).*

11) The method *ST_MultiLineString(ST_LineString ARRAY, INTEGER)* takes the following input parameters:

a) an *ST_LineString* ARRAY value *alinestringarray*,

b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_MultiLineString(ST_LineString ARRAY, INTEGER)* returns an *ST_MultiLineString* value with:

a) The spatial reference system identifier set to *ansrid*.

b) Using the method *ST_Geometries(ST_Geometry ARRAY)*:

i) the *ST_PrivateDimension* attribute set to 1 (one).

ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(alinestringarray).*

iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(alinestringarray)*.

iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(alinestringarray)*.

v) the *ST_PrivateGeometries* attribute set to *alinestringarray*.

10.4.3 ST_Geometries Methods

Purpose

Observe and mutate the ST_PrivateGeometries attribute of an ST_MultiLineString value.

Definition

```
CREATE METHOD ST_Geometries()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_MultiLineString
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateGeometries AS
               ST_LineString ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_Geometries
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiLineString
   FOR ST_MultiLineString
   BEGIN
      DECLARE alinestringarray ST_LineString
         ARRAY[ST_MaxGeometryArrayElements];

      -- Cast ageometryarray to an ST_LineString ARRAY
      SET alinestringarray = CAST(ageometryarray AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]);
      -- If SELF is the null value, then return the null value. Otherwise,
      -- return an ST_MultiLineString value containing alinestringarray.
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               (SELF AS ST_MultiCurve).
                  ST_Geometries(alinestringarray)
         END;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Geometries()* has no input parameters.

2) For the null-call method *ST_Geometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the value of the *ST_PrivateGeometries* attribute as an *ST_LineString ARRAY*.

3) The method *ST_Geometries(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry ARRAY* value *ageometryarray*.

4) For the type-preserving method *ST_Geometries(ST_Geometry ARRAY)*:

   a) Let *alinestringarray* be the result of casting *ageometryarray* to an *ST_LineString* ARRAY value (implicitly using *ST_ToLineStringAry(ST_Geometry ARRAY)*).

b) Case:

   i) If SELF is the null value, then return the null value.

   ii) Otherwise, return an *ST_MultiLineString* value with:

      1) The dimension set to 1 (one).

      2) Using the method *ST_Geometries(ST_Geometry ARRAY)* for type *ST_MultiCurve*, the *ST_PrivateGeometries* attribute set to *alinestringarray*.

10.4.4 ST_MLineFromText Functions

Purpose

Return an ST_MultiLineString value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_MLineFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MLineFromText(awkt, 0)

CREATE FUNCTION ST_MLineFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_MLineFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_MLineFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MLineFromText(awkt, 0)*.
3) The function *ST_MLineFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MLineFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_MultiLineString* value.

      If *awkt* is not producible in the BNF for <multilinestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_MultiLineString)*.

10.4.5 ST_MLineFromWKB Functions

Purpose

Return an ST_MultiLineString value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_MLineFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MLineFromWKB(awkb, 0)

CREATE FUNCTION ST_MLineFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_MLineFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_MLineFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_MLineFromWKB(awkb, 0)*.
3) The function *ST_MLineFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MLineFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiLineString* value.

      If *awkb* is not producible in the BNF for <multilinestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_MultiLineString)*.

10.4.6 ST_MLineFromGML Functions

Purpose

Return an ST_MultiLineString value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representationof an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_MLineFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MLineFromGML(agml, 0)

CREATE FUNCTION ST_MLineFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_MultiLineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_MLineFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_MLineFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MLineFromGML(agml, 0)*.
3) The function *ST_MLineFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MLineFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a MultiLineString XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_MultiLineString)*.

10.5 ST_MultiSurface Type and Routines

10.5.1 ST_MultiSurface Type

Purpose

The ST_MultiSurface type is a 2-dimensional geometry and represents a collection of ST_Surface values.

Definition

```
CREATE TYPE ST_MultiSurface
   UNDER ST_GeomCollection
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_MultiSurface
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_MultiSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiSurface
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_MultiSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiSurface
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_MultiSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiSurface
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_MultiSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiSurface
      (asurfacearray ST_Surface
         ARRAY[ST_MaxGeometryArrayElements])
      RETURNS ST_MultiSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_MultiSurface
   (asurfacearray ST_Surface
      ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Area()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Area
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DArea()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DArea
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Perimeter()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Perimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))RETURNS DOUBLE
   PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_3DPerimeter()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DPerimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))RETURNS DOUBLE
   PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Centroid()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DCentroid()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PointOnSurface()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_3DPointOnSurf()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_Geometries()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_Geometries
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiSurface
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The *ST_MultiSurface* type provides for public use:
   a) a method *ST_MultiSurface(CHARACTER LARGE OBJECT)*,
   b) a method *ST_MultiSurface(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_MultiSurface(BINARY LARGE OBJECT)*,
   d) a method *ST_MultiSurface(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_MultiSurface(ST_Surface ARRAY)*,
   f) a method *ST_MultiSurface(ST_Surface ARRAY, INTEGER)*,
   g) a method *ST_Area()*,
   h) a method *ST_Area(CHARACTER VARYING)*,
   i) a method *ST_3DArea()*,
   j) a method *ST_3DArea(CHARACTER VARYING)*,
   k) a method *ST_Perimeter()*,
   l) a method *ST_Perimeter(CHARACTER VARYING)*,
   m) a method *ST_3DPerimeter()*,
   n) a method *ST_3DPerimeter(CHARACTER VARYING)*,
   o) a method *ST_Centroid()*,
   p) a method *ST_3DCentroid()*,
   q) a method *ST_PointOnSurface()*,
   r) a method *ST_3DPointOnSurf()*,
   s) an overriding method *ST_Geometries()*,
   t) an overriding method *ST_Geometries(ST_Geometry ARRAY)*,
   u) a function *ST_MSurfaceFromTxt(CHARACTER LARGE OBJECT)*,
   v) a function *ST_MSurfaceFromTxt(CHARACTER LARGE OBJECT, INTEGER)*,
   w) a function *ST_MSurfaceFromWKB(BINARY LARGE OBJECT)*,
   x) a function *ST_MSurfaceFromWKB(BINARY LARGE OBJECT, INTEGER)*,
   y) a function *ST_MSurfaceFromGML(CHARACTER LARGE OBJECT)*,
   z) a function *ST_MSurfaceFromGML(CHARACTER LARGE OBJECT, INTEGER)*.
2) The dimension of an *ST_MultiSurface* value is 2.
3) The interiors of any two *ST_Surface* values in an *ST_MultiSurface* shall not spatially intersect. The boundaries of any two coplanar elements in the *ST_MultiSurface* shall, at most, intersect at a finite number of points.

   NOTE If they were to meet along a curve, they could be merged into a single surface.
4) An *ST_MultiSurface* value is simple.
5) An *ST_MultiSurface* value is well formed only if all of the *ST_Surface* values in the *ST_PrivateGeometries* attribute are well formed.
6) An *ST_MultiSurface* value returned by the constructor function corresponds to the empty set.

10.5.2 ST_MultiSurface Methods

Return an ST_MultiSurface value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Surface values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_MultiSurface
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   RETURN NEW ST_MultiSurface(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_MultiSurface
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_MultiSurface
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   RETURN NEW ST_MultiSurface(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_MultiSurface
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   RETURN ST_MSurfaceFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_MultiSurface
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   RETURN SELF.ST_SRID(0).ST_Geometries(asurfacearray)

CREATE CONSTRUCTOR METHOD ST_MultiSurface
   (asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(asurfacearray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_MultiSurface(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_MultiSurface(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiSurface(awktorgml, 0)*.

3) The method *ST_MultiSurface(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_MultiSurface(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a MultiSurface XML element in the GML representation, then return the result of the value expression: *ST_MSurfaceFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_MSurfaceFromTxt(awktorgml, ansrid)*.

5) The method *ST_MultiSurface(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_MultiSurface(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiSurface(awkb, 0)*.

7) The method *ST_MultiSurface(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_MultiSurface(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_MSurfaceFromWKB(awkb, ansrid)*.

9) The method *ST_MultiSurface(ST_Surface ARRAY)* takes the following input parameters:

   a) an *ST_Surface* ARRAY value *asurfacearray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_MultiSurface(ST_Surface ARRAY)* returns the result of the value expression: *NEW ST_MultiSurface(asurfacearray, 0).*

11) The method *ST_MultiSurface(ST_Surface ARRAY, INTEGER)* takes the following input parameters:

   a) an *ST_Surface* ARRAY value *asurfacearray*,

   b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_MultiSurface(ST_Surface ARRAY, INTEGER)* returns an *ST_MultiSurface* value with:

   a) The spatial reference system identification set to *ansrid*.

   b) Using the method *ST_Geometries(ST_Geometry ARRAY)*:

      i) the *ST_PrivateDimension* attribute set to 2.

      ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(asurfacearray).*

      iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(asurfacearray)*.

      iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(asurfacearray)*.

      v) the *ST_PrivateGeometries* attribute set to *asurfacearray*.

10.5.3 ST_Area Methods

Purpose

Return the area measurement of an ST_MultiSurface value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Area()
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE area DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET area = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET area = area + SELF.ST_GeometryN(counter).ST_Area();
         SET counter = counter + 1;
      END WHILE;
      RETURN area;
   END

CREATE METHOD ST_Area
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE area DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET area = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET area = area + SELF.ST_GeometryN(counter).ST_Area(aunit);
         SET counter = counter + 1;
      END WHILE;
      RETURN area;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_Area()* has no input parameters.
2) For the null-call method *ST_Area()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the sum of the *ST_Area()* values of the elements in the *ST_PrivateGeometries* attribute of SELF.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Area()* is in the linear unit of measure identified by <linear unit> squared.

ii) Otherwise, the value returned by *ST_Area()* is in an implementation-defined unit of measure.

3) The method *ST_Area(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_Area(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute sum of the *ST_Area(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the sum of the *ST_Area(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF.

e) The returned value is in the units indicated by *aunit*.

5) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

10.5.4 ST_3DArea Methods

Purpose

Return the area measurement of an ST_MultiSurface value, considering z coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DArea()
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE area DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET area = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET area = area + SELF.ST_GeometryN(counter).ST_3DArea();
         SET counter = counter + 1;
      END WHILE;
      RETURN area;
   END

CREATE METHOD ST_3DArea
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE area DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET area = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET area = area + SELF.ST_GeometryN(counter).ST_3DArea(aunit);
         SET counter = counter + 1;
      END WHILE;
      RETURN area;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_3DArea()* has no input parameters.
2) For the null-call method *ST_3DArea()*:
   a) Case:
      i) If SELF is an empty set, then return the null value.
      ii) Otherwise, return the sum of the *ST_3DArea()* values of the elements in the *ST_PrivateGeometries* attribute of SELF.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DArea()* is in the linear unit of measure identified by <linear unit> squared.

ii) Otherwise, the value returned by *ST_3DArea()* is in an implementation-defined unit of measure.

3) The method *ST_3DArea(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_3DArea(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute sum of the *ST_3DArea(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the sum of the *ST_3DArea(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF.

e) The returned value is in the units indicated by *aunit*.

5) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.

10.5.5 ST_Perimeter Methods

Purpose

Return the length measurement of the boundary of an ST_MultiSurface value, ignoring z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_Perimeter()
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE perimeter DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET perimeter = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET perimeter = perimeter +
            SELF.ST_GeometryN(counter).ST_Perimeter();
         SET counter = counter + 1;
      END WHILE;
      RETURN perimeter;
   END

CREATE METHOD ST_Perimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE perimeter DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET perimeter = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET perimeter = perimeter +
            SELF.ST_GeometryN(counter).ST_Perimeter(aunit);
         SET counter = counter + 1;
      END WHILE;
      RETURN perimeter;
   END
```

Description

1) The method *ST_Perimeter()* has no input parameters.

2) For the null-call method *ST_Perimeter()*:

   a) Case:

      i) If SELF is an empty set, then return the null value.

      ii) Otherwise, return the sum of the *ST_Perimeter* value of the elements in the *ST_PrivateGeometries* attribute of SELF.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_Perimeter()* is in the linear unit of measure identified by <linear unit> squared.

ii) Otherwise, the value returned by *ST_Perimeter()* is in an implementation-defined unit of measure.

3) The method *ST_Perimeter(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_Perimeter(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute sum of the *ST_Perimeter(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the sum of the *ST_Perimeter(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF.

e) The returned value is in the units indicated by *aunit*.

5) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are not considered in the calculation.

10.5.6 ST_3DPerimeter Methods

Purpose

Return the length measurement of the boundary of an ST_MultiSurface value, considering z and m coordinate values in the calculations.

Definition

```
CREATE METHOD ST_3DPerimeter()
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE perimeter DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET perimeter = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET perimeter = perimeter +
            SELF.ST_GeometryN(counter).ST_3DPerimeter();
         SET counter = counter + 1;
      END WHILE;
      RETURN perimeter;
   END

CREATE METHOD ST_3DPerimeter
   (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   FOR ST_MultiSurface
   BEGIN
      DECLARE perimeter DOUBLE PRECISION;
      DECLARE counter INTEGER;

      IF SELF.ST_IsEmpty() = 1 THEN
         RETURN CAST (NULL AS DOUBLE PRECISION);
      END IF;
      SET perimeter = 0.0;
      SET counter = 1;
      WHILE counter <= SELF.ST_NumGeometries() DO
         SET perimeter = perimeter +
            SELF.ST_GeometryN(counter).ST_3DPerimeter(aunit);
         SET counter = counter + 1;
      END WHILE;
      RETURN perimeter;
   END
```

Description

1) The method *ST_3DPerimeter()* has no input parameters.

2) For the null-call method *ST_3DPerimeter()*:

   a) Case:

      i) If SELF is an empty set, then return the null value.

      ii) Otherwise, return the sum of the *ST_3DPerimeter* value of the elements in the *ST_PrivateGeometries* attribute of SELF.

b) Case:

i) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_3DPerimeter()* is in the linear unit of measure identified by <linear unit> squared.

ii) Otherwise, the value returned by *ST_3DPerimeter()* is in an implementation-defined unit of measure.

3) The method *ST_3DPerimeter(CHARACTER VARYING)* takes the following input parameter:

a) a CHARACTER VARYING value *aunit*.

4) For the null-call method *ST_3DPerimeter(CHARACTER VARYING)*:

a) The values for *aunit* shall be a supported <unit name>.

b) The value for *aunit* is a supported <unit name> if and only if the value of *aunit* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_UNITS_OF_MEASURE view.

c) If the unit specified by *aunit* is not supported by the implementation to compute sum of the *ST_3DPerimeter(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Case:

i) If SELF is an empty set, then return the null value.

ii) Otherwise, return the sum of the *ST_3DPerimeter(aunit)* values of each element in the *ST_PrivateGeometries* attribute of SELF.

e) The returned value is in the units indicated by *aunit*.

5) If *SELF.ST_Is3D()* is equal to 1 (one), then the z coordinate values are considered in the calculation.

10.5.7 ST_Centroid Method

Purpose

Return the ST_Point value that is the mathematical centroid of the ST_MultiSurface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_Centroid()
   RETURNS ST_Point
   FOR ST_MultiSurface
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Centroid()* has no input parameters.

2) For the null-call method *ST_Centroid()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Return the mathematical centroid of the *ST_MultiSurface* value. The result is not guaranteed to spatially intersect an *ST_Surface* value in the *ST_PrivateGeometries* attribute of an *ST_MultiSurface* value.

      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:

         1) The z coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the z coordinate value.

      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:

         1) The m coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the m coordinate value.

      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

10.5.8 ST_3DCentroid Method

Purpose

Return the ST_Point value that is the mathematical centroid of the ST_MultiSurface value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DCentroid()
   RETURNS ST_Point
   FOR ST_MultiSurface
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DCentroid()* has no input parameters.

2) For the null-call method *ST_3DCentroid()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Return the mathematical centroid of the *ST_MultiSurface* value. The result is not guaranteed to spatially intersect an *ST_Surface* value in the *ST_PrivateGeometries* attribute of an *ST_MultiSurface* value.

      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:

         1) The z coordinate values are considered in the calculation.

         2) The *ST_Point* value includes the z coordinate value.

      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:

         1) The m coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the m coordinate value.

      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

10.5.9 ST_PointOnSurface Method

Purpose

Return an ST_Point value guaranteed to spatially intersect an ST_Surface value in the ST_PrivateGeometries attribute of an ST_MultiSurface value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Definition

```
CREATE METHOD ST_PointOnSurface()
   RETURNS ST_Point
   FOR ST_MultiSurface
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_PointOnSurface()* has no input parameters.

2) For the null-call method *ST_PointOnSurface()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise:

      i) Return an *ST_Point* value guaranteed to spatially intersect an element in the collection of the *ST_MultiSurface* value.

      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:

         1) The z coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the z coordinate value.

      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:

         1) The m coordinate values are not considered in the calculation.

         2) The *ST_Point* value does not include the m coordinate value.

      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

10.5.10 ST_3DPointOnSurf Method

Purpose

Return an ST_Point value guaranteed to spatially intersect an ST_Surface value in the ST_PrivateGeometries attribute of an ST_MultiSurface value, considering z coordinate values in the calculations and including them in the resultant geometry.

Definition

```
CREATE METHOD ST_3DPointOnSurf()
   RETURNS ST_Point
   FOR ST_MultiSurface
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_3DPointOnSurf()* has no input parameters.
2) For the null-call method *ST_PointOnSurf()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise:
      i) Return an *ST_Point* value guaranteed to spatially 3D intersect an element in the collection of the *ST_MultiSurface* value.
      ii) If *SELF.ST_Is3D()* is equal to 1 (one), then:
         1) The z coordinate values are considered in the calculation.
         2) The *ST_Point* value includes the z coordinate value.
      iii) If *SELF.ST_IsMeasured()* is equal to 1 (one), then:
         1) The m coordinate values are not considered in the calculation.
         2) The *ST_Point* value does not include the m coordinate value.
      iv) The spatial reference system identifier of the returned *ST_Geometry* value is equal to the spatial reference system identifier of SELF.

10.5.11 ST_Geometries Methods

Purpose

Observe and mutate the ST_PrivateGeometries attribute of an ST_MultiSurface value.

Definition

```
CREATE METHOD ST_Geometries()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_MultiSurface
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateGeometries AS
               ST_Surface ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_Geometries
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiSurface
   FOR ST_MultiSurface
   BEGIN
      DECLARE acounter INTEGER;
      DECLARE bcounter INTEGER;
      DECLARE asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements];

      -- Cast ageometryarray to an ST_Surface ARRAY
      SET asurfacearray = CAST(ageometryarray AS
         ST_Surface ARRAY[ST_MaxGeometryArrayElements]);
      -- If any two surfaces intersect with the dimension of the result
      -- greater than 0 (zero), then raise an exception.
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(asurfacearray)-1 DO
         SET bcounter = acounter+1;
         WHILE bcounter <= CARDINALITY(asurfacearray) DO
            IF asurfacearray[acounter].ST_Intersection(
               asurfacearray[bcounter]).ST_Dimension() > 0 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            SET bcounter = bcounter + 1;
         END WHILE;
         SET acounter = acounter + 1;
      END WHILE;
      -- If SELF is the null value, then return the null value. Otherwise,
      -- return an ST_MultiSurface value containing asurfacearray.
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               (SELF AS ST_GeomCollection).
                  ST_Geometries(asurfacearray)
         END;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Geometries()* has no input parameters.
2) For the null-call method *ST_Geometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the value of the *ST_PrivateGeometries* attribute as an *ST_Surface* ARRAY.
3) The method *ST_Geometries(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry ARRAY* value *ageometryarray*.
4) For the type-preserving method *ST_Geometries(ST_Geometry ARRAY)*:
   a) Let *asurfacearray* be the result of casting *ageometryarray* to an *ST_Surface* ARRAY value (implicitly using *ST_ToSurfaceAry(ST_Geometry ARRAY)*).
   b) Case:
      i) If any two elements of *asurfacearray* intersect with more than a finite number of points, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
      ii) If SELF is the null value, then return the null value.
      iii) Otherwise, return an *ST_MultiSurface* value with:
         1) The dimension set to 2.
         2) Using the method *ST_Geometries(ST_Geometry ARRAY)* for type *ST_GeomCollection*, the *ST_PrivateGeometries* attribute set to *asurfacearray*.

10.5.12 ST_MSurfaceFromTxt Functions

Purpose

Return an ST_MultiSurface value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiSurface value.

Definition

```
CREATE FUNCTION ST_MSurfaceFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MSurfaceFromText(awkt, 0)

CREATE FUNCTION ST_MSurfaceFromTxt
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_MSurfaceFromTxt(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_MSurfaceFromTxt(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MSurfaceFromTxt(awkt, 0)*.
3) The function *ST_MSurfaceFromTxt(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MSurfaceFromTxt(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_MultiSurface* value.

      If *awkt* is not producible in the BNF for <multisurface text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_MultiSurface)*.

10.5.13 ST_MSurfaceFromWKB Functions

Purpose

Return an ST_MultiSurface value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiSurface value.

Definition

```
CREATE FUNCTION ST_MSurfaceFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MSurfaceFromWKB(awkb, 0)

CREATE FUNCTION ST_MSurfaceFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_MSurfaceFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*.

2) The null-call function *ST_MSurfaceFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_MSurfaceFromWKB(awkb, 0)*.

3) The function *ST_MSurfaceFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

4) For the null-call function *ST_MSurfaceFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiSurface* value.

      If *awkb* is not producible in the BNF for <multisurface binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_MultiSurface)*.

10.5.14 ST_MSurfaceFromGML Functions

Purpose

Return an ST_MultiSurface value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representationof an ST_MultiSurface value.

Definition

```
CREATE FUNCTION ST_MSurfaceFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MSurfaceFromGML(agml, 0)

CREATE FUNCTION ST_MSurfaceFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_MultiSurface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_MSurfaceFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_MSurfaceFromGML*(CHARACTER LARGE OBJECT) returns the result of the value expression: *ST_MSurfaceFromGML(agml, 0)*.
3) The function *ST_MSurfaceFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MSurfaceFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If the parameter *agml* does not contain a MultiSurface XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_MultiSurface)*.

10.6 ST_MultiPolygon Type and Routines

10.6.1 ST_MultiPolygon Type

Purpose

The ST_MultiPolygon type is a subtype of the ST_MultiSurface and represents a collection of ST_Polygon values.

Definition

```
CREATE TYPE ST_MultiPolygon
   UNDER ST_MultiSurface
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_MultiPolygon
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
      RETURNS ST_MultiPolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPolygon
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
       ansrid INTEGER)
      RETURNS ST_MultiPolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPolygon
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
      RETURNS ST_MultiPolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_MultiPolygon
      (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
       ansrid INTEGER)
      RETURNS ST_MultiPolygon
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_MultiPolygon
   (apolygonarray ST_Polygon
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_MultiPolygon
   (apolygonarray ST_Polygon
      ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

OVERRIDING METHOD ST_Geometries()
   RETURNS ST_Polygon ARRAY[ST_MaxGeometryArrayElements],

OVERRIDING METHOD ST_Geometries
   (ageometryarray ST_Geometry
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPolygon
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The *ST_MultiPolygon* type provides for public use:

   a) a method *ST_MultiPolygon(CHARACTER LARGE OBJECT)*,

   b) a method *ST_MultiPolygon(CHARACTER LARGE OBJECT, INTEGER)*,

   c) a method *ST_MultiPolygon(BINARY LARGE OBJECT)*,

   d) a method *ST_MultiPolygon(BINARY LARGE OBJECT, INTEGER)*,

   e) a method *ST_MultiPolygon(ST_Polygon ARRAY)*,

   f) a method *ST_MultiPolygon(ST_Polygon ARRAY, INTEGER)*,

   g) an overriding method *ST_Geometries()*,

   h) an overriding method *ST_Geometries(ST_Geometry ARRAY)*,

   i) a function *ST_MPolyFromText(CHARACTER LARGE OBJECT)*,

   j) a function *ST_MPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*,

   k) a function *ST_MPolyFromWKB(BINARY LARGE OBJECT)*,

   l) a function *ST_MPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*,

   m) a function *ST_MPolyFromGML(CHARACTER LARGE OBJECT)*,

n) a function *ST_MPolyFromGML(CHARACTER LARGE OBJECT, INTEGER)*,

o) a function *ST_BdMPolyFromText(CHARACTER LARGE OBJECT)*,

p) a function ST*_BdMPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*,

q) a function *ST_BdMPolyFromWKB(BINARY LARGE OBJECT)*,

r) a function *ST_BdMPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*.

2) The elements of the *ST_PrivateGeometries* attribute are restricted to *ST_Polygon* values.

3) The interiors of any two *ST_Polygon* values that are elements of the *ST_PrivateGeometries* attribute shall not spatially intersect.

$\forall$ m $\in$ *ST_MultiPolygon*, $\forall$ p_i, p_j $\in$ m.*ST_Geometries()*, $i \neq j$, Interior(p_i) $\cap$ Interior(p_j) = $\varnothing$

4) The boundaries of any two *ST_Polygon* values that are coplanar elements of the *ST_PrivateGeometries* attribute may only intersect at a finite number of points.

$\forall$ m $\in$ *ST_MultiPolygon*, $\forall$ p_i, p_j $\in$ m.*ST_Geometries()* p_i, p_j coplanar,
$\forall$ c_i $\in$ Boundary(p_i), c_j $\in$ Boundary(p_j) $c_i \cap c_j = \{ p_1, ..., p_k \mid p_i \in$ *ST_Point*, $1 \leq i \leq k \}$

5) An *ST_MultiPolygon* is a topologically closed point set.

6) An *ST_MultiPolygon* shall not have cut lines, spikes or punctures.

$\forall$ m $\in$ *ST_MultiPolygon*, m = Closure(Interior(m))

7) The interior of an *ST_MultiPolygon* with more than one *ST_Polygon* value is not a connected point set. The number of connected components of the interior of an *ST_MultiPolygon* is equal to the cardinality of the *ST_PrivateGeometries* attribute.

8) The boundary of an *ST_MultiPolygon* is a set of linear rings corresponding to the boundaries of the *ST_Polygon* values of the *ST_PrivateGeometries*. Each linear ring in the boundary of the *ST_MultiPolygon* is in the boundary of exactly one *ST_Polygon* in the *ST_PrivateGeometries* attribute. Every linear ring in the boundary of an *ST_Polygon* in the *ST_PrivateGeometries* attribute is in the boundary of the *ST_MultiPolygon*.

9) An *ST_MultiPolygon* value is well formed only if and only if all of the *ST_Polygon* values in the *ST_PrivateGeometries* attribute are well formed.

10) An *ST_MultiPolygon* value returned by the constructor function corresponds to the empty set.

10.6.2 ST_MultiPolygon Methods

Purpose

Return an ST_MultiPolygon value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified ST_Polygon values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_MultiPolygon
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   RETURN NEW ST_MultiPolygon(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_MultiPolygon
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_MultiPolygon
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   RETURN NEW ST_MultiPolygon(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_MultiPolygon
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   RETURN ST_MPolyFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_MultiPolygon
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   RETURN SELF.ST_SRID(0).ST_Geometries(apolygonarray)

CREATE CONSTRUCTOR METHOD ST_MultiPolygon
   (apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements],
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   RETURN SELF.ST_SRID(ansrid).ST_Geometries(apolygonarray)
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.
3) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of an *ST_Geometry* value.

Description

1) The method *ST_MultiPolygon(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_MultiPolygon(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiPolygon(awktorgml, 0)*.

3) The method *ST_MultiPolygon(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_MultiPolygon(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a MultiPolygon XML element in the GML representation, then return the result of the value expression: *ST_MPolyFromGML(awktorgml, ansrid)*.

   b) Otherwise, return the result of the value expression: *ST_MPolyFromText(awktorgml, ansrid)*.

5) The method *ST_MultiPolygon(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_MultiPolygon(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_MultiPolygon(awkb, 0)*.

7) The method *ST_MultiPolygon(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_MultiPolygon(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_MPolygonFromWKB(awkb, ansrid)*.

9) The method *ST_MultiPolygon(ST_Polygon ARRAY)* takes the following input parameters:

   a) an *ST_Polygon* value *apolygonarray*.

10) The null-call type-preserving SQL-invoked constructor method *ST_MultiPolygon(ST_Polygon ARRAY)* returns the result of the value expression: *NEW ST_MultiPolygon(apolygonarray, 0).*

11) The method *ST_MultiPolygon(ST_Polygon ARRAY, INTEGER)* takes the following input parameters:

   a) an *ST_Polygon* ARRAY value *apolygonarray*,

   b) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_MultiPolygon(ST_Polygon ARRAY, INTEGER)* returns an *ST_MultiPolygon* value with:

   a) The spatial reference system identifier set to *ansrid*.

   b) Using the method *ST_Geometries(ST_Geometry ARRAY)*:

      i) the *ST_PrivateDimension* attribute set to 2.

      ii) the *ST_PrivateCoordinateDimension* attribute set to the value expression: *ST_GetCoordDim(apolygonarray).*

      iii) the *ST_PrivateIs3D* attribute set to the value expression: *ST_GetIs3D(apolygonarray)*.

      iv) the *ST_PrivateIsMeasured* attribute set to the value expression: *ST_GetIsMeasured(apolygonarray)*.

      v) the *ST_PrivateGeometries* attribute set to *apolygonarray*.

10.6.3 ST_Geometries Methods

Purpose

Observe and mutate the ST_PrivateGeometries attribute of an ST_MultiPolygon value.

Definition

```
CREATE METHOD ST_Geometries()
   RETURNS ST_Polygon ARRAY[ST_MaxGeometryArrayElements]
   FOR ST_MultiPolygon
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CAST(SELF.ST_PrivateGeometries AS
               ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
      END

CREATE METHOD ST_Geometries
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiPolygon
   FOR ST_MultiPolygon
   BEGIN
      DECLARE acounter INTEGER;
      DECLARE bcounter INTEGER;
      DECLARE apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements];

      -- Cast ageometryarray to an ST_Polygon ARRAY
      SET apolygonarray = CAST(ageometryarray AS
         ST_Polygon ARRAY[ST_MaxGeometryArrayElements]);
      -- If any two polygons intersect with the dimension of the result
      -- greater than 0 (zero), then raise an exception.
      SET acounter = 1;
      WHILE acounter <= CARDINALITY(apolygonarray)-1 DO
         SET bcounter = acounter+1;
         WHILE bcounter <= CARDINALITY(apolygonarray) DO
            IF apolygonarray[acounter].ST_Intersection(
               apolygonarray[bcounter]).ST_Dimension() > 0 THEN
               SIGNAL SQLSTATE '2FF02'
                  SET MESSAGE_TEXT = 'invalid argument';
            END IF;
            SET bcounter = bcounter + 1;
         END WHILE;
         SET acounter = acounter + 1;
      END WHILE;
      -- If SELF is the null value, then return the null value. Otherwise,
      -- return an ST_MultiPolygon value containing apolygonarray.
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               (SELF AS ST_MultiSurface).
                  ST_Geometries(apolygonarray)
         END;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The method *ST_Geometries()* has no input parameters.
2) For the null-call method *ST_Geometries()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the value of the *ST_PrivateGeometries* attribute as an *ST_Polygon* ARRAY.
3) The method *ST_Geometries(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
4) For the type-preserving method *ST_Geometries(ST_Geometry ARRAY)*:
   a) Let *apolygonarray* be the result of casting *ageometryarray* to an *ST_Polygon* ARRAY value (implicitly using *ST_ToPolygonAry(ST_Geometry ARRAY*).
   b) Case:
      i) If any two elements of *apolygonarray* intersect with more than a finite number of points, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
      ii) If SELF is the null value, then return the null value.
      iii) Otherwise, return an *ST_MultiPolygon* value with:
         1) The dimension set to 2.
         2) Using the method *ST_Geometries(ST_Geometry ARRAY)* for type *ST_MultiSurface*, the *ST_PrivateGeometries* attribute set to *apolygonarray*.

10.6.4 ST_MPolyFromText Functions

Purpose

Return an ST_MultiPolygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiPolygon value.

Definition

```
CREATE FUNCTION ST_MPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MPolyFromText(awkt, 0)

CREATE FUNCTION ST_MPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Geometry* value.

Description

1) The function *ST_MPolyFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_MPolyFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MPolyFromText(awkt, 0)*.
3) The function *ST_MPolyFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) The parameter *awkt* is the well-known text representation of an *ST_MultiPolygon* value.

      If *awkt* is not producible in the BNF for <multipolygon text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromText(awkt, ansrid) AS ST_MultiPolygon)*.

10.6.5 ST_MPolyFromWKB Functions

Purpose

Return an ST_MultiPolygon value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiPolygon value.

Definition

```
CREATE FUNCTION ST_MPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MPolyFromWKB(awkb, 0)

CREATE FUNCTION ST_MPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

Description

1) The function *ST_MPolyFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_MPolyFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_MPolyFromWKB(awkb, 0)*.
3) The function *ST_MPolyFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiPolygon* value.

      If *awkb* is not producible in the BNF for <multipolygon binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromWKB(awkb, ansrid) AS ST_MultiPolygon)*.

10.6.6 ST_MPolyFromGML Functions

Purpose

Return an ST_MultiPolygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representationof an ST_MultiPolygon value.

Definition

```
CREATE FUNCTION ST_MPolyFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML))
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_MPolyFromGML(agml, 0)

CREATE FUNCTION ST_MPolyFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxGeometryAsGML),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

Description

1) The function *ST_MPolyFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_MPolyFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_MPolyFromGML(agml, 0)*.
3) The function *ST_MPolyFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_MPolyFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a MultiPolygon XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_GeomFromGML(agml, ansrid) AS ST_MultiPolygon)*.

10.6.7 ST_BdMPolyFromText Functions

Purpose

Return an ST_MultiPolygon value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_BdMPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BdMPolyFromText(awkt, 0)

CREATE FUNCTION ST_BdMPolyFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The function *ST_BdMPolyFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_BdMPolyFromText(CHARACTER LARGE OBJECT)* returns an *ST_MultiPolygon* value as the result of the value expression: *ST_BdMPolyFromText(awkt, 0)*.
3) The function *ST_BdMPolyFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BdMPolyFromText(CHARACTER LARGE OBJECT, INTEGER)*:
   a) The parameter *awkt* is the well-known text representation of an *ST_MultiLineString* value. If *awkt* is not producible in the BNF for <multilinestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Use *ST_MLineFromText(CHARACTER LARGE OBJECT)* to transform *awkt* to an *ST_MultiLineString* value, *AMLS.*
   c) If any *ST_LineString* value in *AMLS* is not a linear ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   d) Using an implementation-dependent algorithm, an array of *ST_Polygon* values, *APA*, is determined from the array of linear rings in *AMLS*.
   e) Return an *ST_MultiPolygon* value with:
      i) The spatial reference system identifier set to *ansrid*.

ii) Using the method *ST_Geometries(ST_Geometry ARRAY),* the *ST_PrivateGeometries* attribute set to *APA*.

10.6.8 ST_BdMPolyFromWKB Functions

Purpose

Return an ST_MultiPolygon value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_MultiLineString value.

Definition

```
CREATE FUNCTION ST_BdMPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_BdMPolyFromWKB(awkb, 0)

CREATE FUNCTION ST_BdMPolyFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary),
    ansrid INTEGER)
   RETURNS ST_MultiPolygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The function *ST_BdMPolyFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) For the null-call function *ST_BdMPolyFromWKB(BINARY LARGE OBJECT)* returns an *ST_MultiPolygon* value as the result of the value expression: *ST_BdMPolyFromWKB(awkt, 0)*.
3) The function *ST_BdMPolyFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) an BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_BdMPolyFromWKB(BINARY LARGE OBJECT, INTEGER)*:
   a) The parameter *awkb* is the well-known binary representation of an *ST_MultiLineString* value. If *awkb* is not producible in the BNF for <multilinestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Use *ST_MLineFromWKB(BINARY LARGE OBJECT)* to transform *awkb* to an *ST_MultiLineString* value, *AMLS.*
   c) If any *ST_LineString* value in *AMLS* is not a linear ring, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   d) Using an implementation-dependent algorithm, an array of *ST_Polygon* values, *APA*, is determined from the array of linear rings in *AMLS*.
   e) Return an *ST_MultiPolygon* value with:
      i) The spatial reference system identifier set to *ansrid*.

ii) Using the method *ST_Geometries(ST_Geometry ARRAY)*, the *ST_PrivateGeometries* attribute set to *APA*.

11 Topology-Geometry

11.1 Topo-Geo Topology Schema

11.1.1 Introduction

The Topo-Geo Topology Schema views are defined as being in a schema named <topology-name> enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views may be granted to individual users so that they can be queried. To update a Topo-Geo, a user shall be granted SELECT privlilege on the views for the Topo-Geo and EXECUTE privilege on the Topo-Geo routines provided. Roles can be used to enable update on a selective set of Topo-Geos. These views are updated only by the topology functions provided.

An implementation may define objects that are associated with <topology-name> that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

All of the topological primitives contained in the views owned by the <topology-name> schema constitute a single, topologically consistent, topology representing a planar graph. Other topologies (e.g., covering a different spatial extent, existing at a different level of abstraction, or associated with a different set of features) can exist in another, independent <topology-name> schema.

A face having a FACE_ID equal to 0 (zero) shall exist for every Topo-Geo topology schema. Representing the universal face, it has no associated geometry.

11.1.2 ST_NODE view

Purpose

Contains the node type of topological primitives (ST_Node) contained in the <topology-name> Topo-Geo.

Definition

```
CREATE VIEW ST_NODE AS
   SELECT NODE_ID, GEOMETRY, CONTAINING_FACE
      FROM ST_TOPO_GEO.ST_NODE
      WHERE TOPOLOGY = '<topology-name>'
```


11.1.3 ST_EDGE view

Purpose

Contains the edge type of topological primitives (ST_Edge) contained in the <topology-name> Topo-Geo.

Definition

```
CREATE VIEW ST_EDGE AS
   SELECT EDGE_ID, START_NODE, END_NODE,
         NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE,
         LEFT_FACE, RIGHT_FACE, GEOMETRY
      FROM ST_TOPO_GEO.ST_EDGE
      WHERE TOPOLOGY = '<topology-name>'
```

11.1.4 ST_FACE view

Purpose

Contains the face type of topological primitives (ST_Face) contained in the <topology-name> Topo-Geo.

Definition

```
CREATE VIEW ST_FACE AS
   SELECT FACE_ID, MBR
      FROM ST_TOPO_GEO.ST_FACE
      WHERE TOPOLOGY = '<topology-name>'
```

11.2 Topo-Geo Definition Schema

11.2.1 Introduction

The only purpose of this Topo-Geo Definition Schema is to provide a data model to support Topo-Geo views and to assist understanding. The base tables of this Topo-Geo Definition Schema are defined as being in a schema named ST_TOPO_GEO. The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1) The function of the definition is stated.
2) The SQL definition of the object is presented as a <table definition>.
3) An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

11.2.2 ST_NODE base table

Purpose

Contains the node type of topological primitives (ST_Node) contained in topology-geometries.

Definition

```
CREATE TABLE ST_NODE
   (
   TOPOLOGY CHARACTER VARYING(ST_MaxTopologyName),
   NODE_ID INTEGER NOT NULL,
   GEOMETRY ST_Point NOT NULL,
   CONTAINING_FACE INTEGER,

   CONSTRAINT ST_NODE_PRIMARY_KEY PRIMARY KEY(TOPOLOGY, NODE_ID),
   CONSTRAINT FACE_EXISTS FOREIGN KEY(TOPOLOGY, CONTAINING_FACE)
      REFERENCES ST_FACE(TOPOLOGY, FACE_ID)
   )
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The value of *TOPOLOGY* is the topology name which distinguishes which nodes are in the topology coverage.
2) Let *T* be the value of *TOPOLOGY* for a given node.
3) The value of *NODE_ID* is the identifier of the node unique among all nodes with a *TOPOLOGY* value equal to *T*.
4) The value of *GEOMETRY* is the spatial location of the node.
5) The value of *CONTAINING_FACE* is the unique identifier of the face containing the node if the node is an isolated node; the face shall have a value of *TOPOLOGY* equal to *T*. Otherwise *CONTAINING_FACE* is the null value.
6) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.2.3 ST_EDGE base table

Purpose

Contains the edge type of topological primitives (ST_Edge) contained in topology-geometries.

Definition

```
CREATE TABLE ST_EDGE
   (
   TOPOLOGY CHARACTER VARYING(ST_MaxTopologyName),
   EDGE_ID INTEGER NOT NULL,
   START_NODE INTEGER NOT NULL,
   END_NODE INTEGER NOT NULL,
   NEXT_LEFT_EDGE INTEGER NOT NULL,
   NEXT_RIGHT_EDGE INTEGER NOT NULL,
   LEFT_FACE INTEGER NOT NULL,
   RIGHT_FACE INTEGER NOT NULL,
   GEOMETRY ST_Curve NOT NULL,

   CONSTRAINT ST_EDGE_PRIMARY_KEY PRIMARY KEY(TOPOLOGY, EDGE_ID),
   CONSTRAINT START_NODE_EXISTS FOREIGN KEY(TOPOLOGY, START_NODE)
      REFERENCES ST_NODE(TOPOLOGY, NODE_ID),
   CONSTRAINT END_NODE_EXISTS FOREIGN KEY(TOPOLOGY, END_NODE)
      REFERENCES ST_NODE(TOPOLOGY, NODE_ID),
   CONSTRAINT NEXT_LEFT_EDGE_EXISTS
      CHECK (TOPOLOGY, ABS(NEXT_LEFT_EDGE)) IN
         (SELECT TOPOLOGY, EDGE_ID FROM ST_EDGE)),
   CONSTRAINT NEXT_RIGHT_EDGE_EXISTS
      CHECK (TOPOLOGY, ABS(NEXT_RIGHT_EDGE)) IN
         (SELECT TOPOLOGY, EDGE_ID FROM ST_EDGE)),
   CONSTRAINT LEFT_FACE_EXISTS FOREIGN KEY(TOPOLOGY, LEFT_FACE)
      REFERENCES ST_FACE(TOPOLOGY, FACE_ID),
   CONSTRAINT RIGHT_FACE_EXISTS FOREIGN KEY(TOPOLOGY, RIGHT_FACE)
      REFERENCES ST_FACE(TOPOLOGY, FACE_ID)
   )
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The value of *TOPOLOGY* is the topology name which distinguishes which edges are in the topology coverage.
2) Let *T* be the value of *TOPOLOGY* for a given edge.
3) The value of *EDGE_ID* is the identifier of the edge unique among all edges with a *TOPOLOGY* value equal to *T*.
4) The value of *START_NODE* is the unique identifier of the node at the start of the edge; the start node shall have a value of *TOPOLOGY* equal to *T*.
5) The value of *END_NODE* is the unique identifier of the node at the end of the edge; the end node shall have a value of *TOPOLOGY* equal to *T*.
6) The value of *NEXT_LEFT_EDGE* is the unique identifier of the next edge of the face on the left (when looking in the direction from *START_NODE* to *END_NODE*), moving counterclockwise around the face boundary; the next left edge node shall have a value of *TOPOLOGY* equal to *T*.
7) The value of *NEXT_RIGHT_EDGE* is the unique identifier of the next edge of the face on the right (when looking in the direction from *START_NODE* to *END_NODE*), moving counterclockwise around the face boundary; the next right edge shall have a value of *TOPOLOGY* equal to *T*.

8) The value of *LEFT_FACE* is the unique identifier of the face on the left side of the edge when looking in the direction from *START_NODE* to *END_NODE*; the left face shall have a value of *TOPOLOGY* equal to *T*.

9) The value of *RIGHT_FACE* is the unique identifier of the face on the right side of the edge when looking in the direction from *START_NODE* to *END_NODE*; the right face shall have a value of *TOPOLOGY* equal to *T*.

10) The value of *GEOMETRY* is the geometry of the edge.

11) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.2.4 ST_FACE base table

Purpose

Contains the face type of topological primitives (ST_Face) contained in topology-geometries.

Definition

```
CREATE TABLE ST_FACE
   (
   TOPOLOGY CHARACTER VARYING(ST_MaxTopologyName),
   FACE_ID INTEGER NOT NULL,
   MBR ST_Polygon,

   CONSTRAINT ST_FACE_PRIMARY_KEY PRIMARY KEY(TOPOLOGY, FACE_ID)
   )
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The value of *TOPOLOGY* is the topology name which distinguishes which faces are in the topology coverage.
2) Let *T* be the value of *TOPOLOGY* for a given face.
3) The value of *FACE_ID* is the identifier of the face unique among all faces with a *TOPOLOGY* value equal to *T*.
4) The value of *MBR* is the geometry of the minimum bounding rectangle of the face.
5) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3 Topo-Geo Routines

11.3.1 ST_AddIsoNode Function

Purpose

Insert a row into the <topology-name>.*ST_NODE* view for an isolated node.

Definition

```
CREATE FUNCTION ST_AddIsoNode
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    aface INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddIsoNode(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *aface*,
   c) an *ST_Point* value *apoint*.
2) For the function *ST_AddIsoNode(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If any *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.

ii) Let *E1* be an INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*. Let *G1* be an *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E1*. If the value returned by *G1.ST_Crosses(apoint)* is equal to 1 (one) for any value of *E1*, then an exception condition is raised: *SQL/MM Spatial exception – edge crosses node.*

iii) Let *F* be an INTEGER value equal to 0 (zero). Let *F1* be an INTEGER value other than 0 (zero) equal to *ST_TOPO_GEO.ST_FACE.FACE_ID* for a face in *atopology*. Let *G1* be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F1)*. If the value returned by *apoint.ST_Within(G1)* is equal to 1 (one) for any value of *G1*, then set *F* equal to the corresponding value of *F1*. If *aface* is NULL, then set *aface* equal to *F*. Otherwise, if *aface* is not equal to *F*, then an exception condition is raised: *SQL/MM Spatial exception – not within face*.

iv) Otherwise:

1) generate a unique node id INTEGER value *anodeid*.

2) insert into *ST_TOPO_GEO*.*ST_NODE* values (*atopology*, *anodeid*, *aface*, *apoint*).

3) return *anodeid*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.2 ST_MoveIsoNode Procedure

Purpose

Update the <topology-name>.*ST_NODE*.*GEOMETRY* value of an isolated node.

Definition

```
CREATE PROCEDURE ST_MoveIsoNode
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    apoint ST_Point)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_MoveIsoNode(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anode*,
   c) an *ST_Point* value *apoint*.
2) For the procedure *ST_MoveIsoNode(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If any *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.
      ii) If any *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

iii) Let *E1* be an INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*. Let *G1* be an *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E1*. If the value returned by *G1.ST_Crosses(apoint)* is equal to 1 (one) for any value of *E1*, then an exception condition is raised: *SQL/MM Spatial exception – edge crosses node.*

iv) Otherwise, update *ST_TOPO_GEO*.*ST_NODE*, set the *GEOMETRY* value equal to *apoint* where *TOPOLOGY* is equal to *atopology* and *NODE_ID* is equal to *anode*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.3 ST_RemIsoNode Procedure

Purpose

Delete a row in the <topology-name>.*ST_NODE* view corresponding to an isolated node.

Definition

```
CREATE PROCEDURE ST_RemIsoNode
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_RemIsoNode(CHARACTER VARYING, INTEGER)* takes the following input parameters:

   a) a CHARACTER VARYING value *atopology*,

   b) an INTEGER value *anode*.

2) For the procedure *ST_RemIsoNode(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.

   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.

   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:

      i) If any *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

      ii) Otherwise, delete from *ST_TOPO_GEO*.*ST_NODE* where *TOPOLOGY* is equal to *atopology* and *NODE_ID* is equal to *anode*.

11.3.4 ST_AddIsoEdge Function

Purpose

Insert a row for an isolated edge into the <topology-name>.*ST_EDGE* view connecting two existing isolated nodes.

Definition

```
CREATE FUNCTION ST_AddIsoEdge
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddIsoEdge(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anode*,
   c) an INTEGER value *anothernode*,
   d) an ST_Curve value *acurve*.
2) For the function *ST_AddIsoEdge(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   g) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.

h) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

Case:

i) If SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_NODE* WHERE *NODE_ID* IS EQUAL TO *anode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node.*

ii) If SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_NODE* WHERE *NODE_ID* IS EQUAL TO *anothernode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node.*

iii) If any *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

iv) If any *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value is equal to *anothernode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

v) Let *F1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*CONTAINING_FACE* where *NODE_ID* is equal to *anode*. Let *F2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*CONTAINING_FACE* where *NODE_ID* is equal to *anothernode*. If *F1* is not equal to *F2*, then an exception condition is raised: *SQL/MM Spatial exception – nodes in different faces*.

vi) If *acurve.ST_Within(ST_GetFaceGeometry(atopology, F1))* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – geometry not within face*.

vii) Let *P1* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to *acurve.ST_StartPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

viii) Let *P2* be the ST_Point value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

ix) Let *N1* be a *NODE_ID* value in *ST_TOPO_GEO*.*ST_NODE* such that N1 does not exist as either an *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value. Let G1 be the *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

x) Let *G* be an *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* value. If *acurve.ST_Intersects(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry intersects an edge*.

xi) Otherwise:

1) generate a unique edge id INTEGER value *anedgeid*.

2) insert into TOPO_GEO.*ST_EDGE* values (*atopology*, *anedgeid*, *anode*, *anothernode*, -(*anedgeid*), *anedgeid*, *F1*, *F1*, *acurve*).

3) return *anedgeid*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.5 ST_GetFaceEdges Function

Purpose

Return a table containing signed edge IDs for the edges that bound a face, in counterclockwise order.

Definition

```
CREATE FUNCTION ST_GetFaceEdges
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    aface INTEGER)
   RETURNS TABLE
      (sequence INTEGER,
       edge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_GetFaceEdges(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *aface*.
2) For the function *ST_GetFaceEdges(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *aface* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:
      i) Return a table value consisting of the following two columns:
         1) a column *edge* of type INTEGER, which contains the signed edge IDs of the bounding edges of the face having an *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* value equal to *aface*, such that edge IDs for edges having the specified face on the right side of the edge when looking in the direction of the edge from start node to end node are negated.
         2) a column *sequence* containing consecutive values of type INTEGER, which represents the ordering of the edges in column *edge* consistent with a counterclockwise traversal around the face. The row with the lowest value for *edge* shall have a *sequence* value of 1 (one).

11.3.6 ST_ChangeEdgeGeom Procedure

Purpose

Update the <topology-name>.*ST_EDGE*.*GEOMETRY* value.

Definition

```
CREATE PROCEDURE ST_ChangeEdgeGeom
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    acurve ST_Curve)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_ChangeEdgeGeom(CHARACTER VARYING, INTEGER, ST_Curve)* takes the following input parameters:

   a) a CHARACTER VARYING value *atopology*,

   b) an INTEGER value *anedge*,

   c) an *ST_Curve* value *acurve*.

2) For the procedure *ST_ChangeEdgeGeom(CHARACTER VARYING, INTEGER, ST_Curve)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.

   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.

   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   e) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   f) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.

   g) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:

      i) Let *P1* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to the value of *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* where *EDGE_ID* is equal to *anedge*. If *P1* is not equal to *acurve.ST_StartPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

ii) Let *P2* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to the value of *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* where *EDGE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

iii) Let *N1* be a *NODE_ID* value in *ST_TOPO_GEO*.*ST_NODE* such that *N1* does not exist as either an *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value. Let *G1* be the *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

iv) Let *G* be an *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* value. If *acurve.ST_Crosses(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry intersects an edge*.

v) Otherwise, update *ST_TOPO_GEO*.*ST_EDGE*, set the *GEOMETRY* value equal to *acurve* where *TOPOLOGY* is equal to atopology and *EDGE_ID* is equal to *anedge*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.7 ST_RemIsoEdge Procedure

Purpose

Delete a row in the <topology-name>.*ST_EDGE* view corresponding to an isolated edge.

Definition

```
CREATE PROCEDURE ST_RemIsoEdge
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_RemIsoEdge(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*.
2) For the procedure *ST_RemIsoEdge(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If *ST_TOPO_GEO*.*ST_EDGE*.*LEFT_FACE* is not equal to *ST_TOPO_GEO*.*ST_EDGE*.*RIGHT_FACE*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated edge*.
      ii) Let *N1* be the *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* of the edge whose *EDGE_ID* is equal to *anedge*. If any *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value is equal to *N1*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated edge*.
      iii) Let *N2* be the *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* of the edge whose *EDGE_ID* is equal to *anedge*. If any *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value is equal to *N2*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated edge*.

iv) Otherwise, delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge.*

11.3.8 ST_NewEdgesSplit Function

Purpose

Split an edge by creating a new node along an existing edge, deleting the original edge and replacing it with two new edges.

Definition

```
CREATE FUNCTION ST_NewEdgesSplit
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_NewEdgesSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*,
   c) an *ST_Point* value *apoint*.
2) For the function *ST_NewEdgesSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.

ii) Let *G* be equal to the *ST_Curve* value of *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *anedge*. If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on edge*.

iii) If any *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.

iv) Otherwise:

1) generate a unique node id INTEGER value *newnode*.

2) insert into *ST_TOPO_GEO*.*ST_NODE* values (*atopology*, *newnode*, *apoint*, NULL).

3) select from *ST_TOPO_GEO*.*ST_EDGE* *START_NODE*, *END_NODE*, *NEXT_LEFT_EDGE*, *NEXT_RIGHT_EDGE*, *LEFT_FACE*, *RIGHT_FACE*, and *GEOMETRY* into *oldstart*, *oldend*, *oldnextleft*, *oldnextright*, *oldleft*, *oldright*, and *oldgeom* where *EDGE_ID* is equal to *anedge*.

4) delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*.

5) generate two unique edge id INTEGER values *newedge1* and *newedge2.*

6) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*.

7) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge1*, *oldstart*, *newnode*, *newedge2*, *oldnextright*, *oldleft*, *oldright*, *newgeom1*).

8) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge2*, *newnode*, *oldend*, *oldnextleft*, -(*newedge1*), *oldleft*, *oldright*, *newgeom2*).

9) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edge being split.

10) return *newnode*.

v) Both new edges have the same direction as the edge being split.

vi) To determine the two new edge ID values, select *EDGE_ID* from *ST_TOPO_GEO*.*ST_EDGE* where *START_NODE* or *END_NODE* is equal to *newnode*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.9 ST_ModEdgeSplit Function

Purpose

Split an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge.

Definition

```
CREATE FUNCTION ST_ModEdgeSplit
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_ModEdgeSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*,
   c) an *ST_Point* value *apoint*.
2) For the function *ST_ModEdgeSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.

ii) Let *G* be equal to the *ST_Curve* value of *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *anedge*. If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on edge*.

iii) If any *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.

iv) Otherwise:

1) generate a unique node id INTEGER value *newnode*.

2) insert into *ST_TOPO_GEO*.*ST_NODE* values (*atopology*, *newnode*, *apoint*, NULL).

3) select from *ST_TOPO_GEO*.*ST_EDGE* *END_NODE*, *NEXT_LEFT_EDGE*, *LEFT_FACE*, *RIGHT_FACE*, and *GEOMETRY* into *oldend*, *oldnextleft*, *oldleft*, *oldright*, and *oldgeom* where *EDGE_ID* is equal to *anedge*.

4) generate a unique edge id INTEGER value *newedge*.

5) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*.

6) update *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to atopology and *EDGE_ID* is equal to *anedge*:

A) set the *END_NODE* value equal to *newnode*.

B) set the *NEXT_LEFT_EDGE* value equal to *newedge*.

C) set the *GEOMETRY* value equal to *newgeom1*.

7) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge*, *newnode*, *oldend*, *oldnextleft*, -(*anedge*), *oldleft*, *oldright*, *newgeom2*).

8) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edge being split.

9) return *newnode*.

v) The new and modified edges have the same direction as the edge being split.

vi) To determine the new edge ID value, select *EDGE_ID* from *ST_TOPO_GEO*.*ST_EDGE* where *START_NODE* is equal to *newnode*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.10 ST_NewEdgeHeal Function

Purpose

Heal two edges by deleting the node connecting them, deleting both edges, and replacing them with a new edge whose direction is the same as the first edge provided.

Definition

```
CREATE FUNCTION ST_NewEdgeHeal
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    anotheredge INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_NewEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*,
   c) an INTEGER value *anotheredge*.
2) For the function *ST_NewEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anotheredge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:

      i) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.

ii) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.

iii) Check if edges are connected:

1) let *COMMONNODE* be an INTEGER value.

2) let *CASE* be an INTEGER value.

3) let *S1* and *E1* be INTEGER values equal to the *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* and *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* values, respectively, where *EDGE_ID* is equal to *anedge*.

4) let *S2* and *E2* be INTEGER values equal to the *ST_TOPO_GEO*.*ST_EDGE.START_NODE* and *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* values, respectively, where *EDGE_ID* is equal to *anotheredge*.

5) case:

A) if *E1* is equal to *S2*, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to *E1*.

B) if *E1* is equal to *E2*, then set *CASE* equal to 2 and *COMMONNODE* equal to *E1*.

C) if *S1* is equal to *S2*, then set *CASE* equal to 3 and *COMMONNODE* equal to *S1*.

D) if *S1* is equal to *E2*, then set *CASE* equal to 4 and *COMMONNODE* equal to *S1*.

E) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected edges*.

iv) If *COMMONNODE* is equal to the *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value for any edge other than the edge whose *EDGE_ID* is equal to either *anedge* or *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – other edges connected*.

v) Otherwise:

1) delete from *ST_TOPO_GEO*.*ST_NODE* where *TOPOLOGY* is equal to *atopology* and *NODE_ID* is equal to *COMMONNODE*.

2) select from *ST_TOPO_GEO*.*ST_EDGE START_NODE*, *END_NODE*, *NEXT_LEFT_EDGE*, *NEXT_RIGHT_EDGE*, *LEFT_FACE*, *RIGHT_FACE*, and *GEOMETRY* into *oldstart1*, *oldend1*, *oldnextleft1*, *oldnextright1*, *oldleft*, *oldright*, and *oldgeom1* where *EDGE_ID* is equal to *anedge*.

3) select from *ST_TOPO_GEO*.*ST_EDGE START_NODE*, *END_NODE*, *NEXT_LEFT_EDGE*, *NEXT_RIGHT_EDGE*, and *GEOMETRY* into *oldstart2*, *oldend2*, *oldnextleft2*, *oldnextright2*, and *oldgeom2* where *EDGE_ID* is equal to *anotheredge*.

4) delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*.

5) delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anotheredge*.

6) generate a unique edge id INTEGER value *newedge*.

7) case:

A) if *CASE* is equal to 1 (one) then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*.

II) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge*, *oldstart1*, *oldend2*, *oldnextleft2*, *oldnextright1*, *oldleft, oldright*, *newgeom*).

B) if *CASE* is equal to 2 then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*.

II) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge*, *oldstart1*, *oldstart2*, *oldnextright2*, *oldnextright1*, *oldleft*, *oldright*, *newgeom*).

C) if *CASE* is equal to 3 then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end.

II) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge*, *oldend2*, *oldend1*, *oldnextleft1*, *oldnextleft2*, *oldleft*, *oldright*, *newgeom*).

D) if *CASE* is equal to 4 then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*.

II) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *newedge*, *oldstart2*, *oldend1*, *oldnextleft1*, *oldnextright2*, *oldleft*, *oldright*, *newgeom*).

8) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the new edge.

9) return *newedge*.

vi) The direction of the new edge shall be the same as the direction of the first edge supplied.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.11 ST_ModEdgeHeal Procedure

Purpose

Heal two edges by deleting the node connecting them, modifying the first edge provided, and deleting the second edge.

Definition

```
CREATE PROCEDURE ST_ModEdgeHeal
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    anotheredge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_ModEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*,
   c) an INTEGER value *anotheredge*.
2) For the procedure *ST_ModEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anotheredge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
      ii) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.

iii) Check if edges are connected:

1) let *COMMONNODE* be an INTEGER value.

2) let *CASE* be an INTEGER value.

3) let *S1* and *E1* be INTEGER values equal to the *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* and *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* values, respectively, where *EDGE_ID* is equal to *anedge*.

4) let *S2* and *E2* be INTEGER values equal to the *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* and *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* values, respectively, where *EDGE_ID* is equal to *anotheredge*.

5) case:

A) if *E1* is equal to *S2*, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to *E1*.

B) if *E1* is equal to *E2*, then set *CASE* equal to 2 and *COMMONNODE* equal to *E1*.

C) if *S1* is equal to *S2*, then set *CASE* equal to 3 and *COMMONNODE* equal to *S1*.

D) if *S1* is equal to *E2*, then set *CASE* equal to 4 and *COMMONNODE* equal to *S1*.

E) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected edges*.

iv) If *COMMONNODE* is equal to the *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* or *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* value for any edge other than the edge whose *EDGE_ID* is equal to either *anedge* or *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – other edges connected*.

v) Otherwise:

1) delete from *ST_TOPO_GEO*.*ST_NODE* where *TOPOLOGY* is equal to *atopology* and *NODE_ID* is equal to *COMMONNODE*.

2) select from *ST_TOPO_GEO*.*ST_EDGE GEOMETRY* into *oldgeom1* where *EDGE_ID* is equal to *anedge*.

3) select from *ST_TOPO_GEO*.*ST_EDGE START_NODE*, *END_NODE*, *NEXT_LEFT_EDGE*, *NEXT_RIGHT_EDGE*, and *GEOMETRY* into *oldstart2*, *oldend2*, *oldnextleft2*, *oldnextright2*, and *oldgeom2* where *EDGE_ID* is equal to *anotheredge*.

4) delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anotheredge*.

5) case:

A) if *CASE* is equal to 1 (one) then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*.

II) update *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*:

1) set the *END_NODE* value equal to *oldend2*.

2) set the *NEXT_LEFT_EDGE* value equal to *oldnextleft2*.

3) set the *GEOMETRY* value equal to *newgeom*.

B) if *CASE* is equal to 2 then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*.

II) update *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*:

1) set the *END_NODE* value equal to *oldstart2*.

2) set the *NEXT_LEFT_EDGE* value equal to *oldnextright2*.

3) set the *GEOMETRY* value equal to *newgeom*.

C) if *CASE* is equal to 3 then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end.

II) update *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*:

1) set the *START_NODE* value equal to *oldend2*.

2) set the *NEXT_RIGHT_EDGE* value equal to *oldnextleft2*.

3) set the *GEOMETRY* value equal to *newgeom*.

D) if *CASE* is equal to 4 then:

I) create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*.

II) update *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*:

1) set the *START_NODE* value equal to *oldstart2*.

2) set the *NEXT_RIGHT_EDGE* value equal to *oldnextright2*.

3) set the *GEOMETRY* value equal to *newgeom*.

6) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the new edge.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.12 ST_AddEdgeNewFaces Function

Purpose

Add a new edge and, if in doing so it splits a face, delete the original face and replace it with two new faces.

Definition

```
CREATE FUNCTION ST_AddEdgeNewFaces
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddEdgeNewFaces(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anode*,
   c) an INTEGER value *anothernode*,
   d) an *ST_Curve* value *acurve*.
2) For the function *ST_AddEdgeNewFaces(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   g) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.

h) Otherwise, for rows in *ST_TOPO_GEO.ST_NODE*, *ST_TOPO_GEO.ST_EDGE*, and *ST_TOPO_GEO.ST_FACE* where *TOPOLOGY* is equal to *atopology*:

Case:

i) If there is no row in *ST_TOPO_GEO.ST_NODE* with a *NODE_ID* value equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

ii) If there is no row in *ST_TOPO_GEO.ST_NODE* with a *NODE_ID* value equal to *anothernode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

iii) Let *P1* be the *ST_Point* value equal to *ST_TOPO_GEO.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to *acurve.ST_StartPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

iv) Let *P2* be the *ST_Point* value equal to *ST_TOPO_GEO.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

v) Let *N1* be a *NODE_ID* value in *ST_TOPO_GEO.ST_NODE* such that *N1* does not exist as either an *ST_TOPO_GEO.ST_EDGE.START_NODE* or *ST_TOPO_GEO.ST_EDGE.END_NODE* value. Let *G1* be the *ST_TOPO_GEO.ST_NODE.GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

vi) Let *G* be an *ST_TOPO_GEO.ST_EDGE.GEOMETRY* value. If *acurve.ST_Crosses(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses an edge*.

vii) If there exists a row in *ST_TOPO_GEO.ST_EDGE* with a *START_NODE* value equal to *anode* and an *END_NODE* value equal to *anothernode* and a *GEOMETRY* value equal to *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – coincident edge*.

viii) If there exists a row in *ST_TOPO_GEO.ST_EDGE* with a *START_NODE* value equal to *anothernode* and an *END_NODE* value equal to *anode* and a *GEOMETRY* value equal to *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – coincident edge*.

ix) Otherwise:

1) generate a unique edge id INTEGER value *anedgeid*.

2) insert into *ST_TOPO_GEO.ST_EDGE* a new row with:

A) a *TOPOLOGY* value equal to *atopology*.

B) an *EDGE_ID* value equal to *anedgeid*.

C) a *START_NODE* value equal to *anode*.

D) an *END_NODE* value equal to *anothernode*.

E) *NEXT_LEFT_EDGE*, *NEXT_RIGHT_EDGE*, *LEFT_FACE*, and *RIGHT_FACE* values as appropriate.

F) a *GEOMETRY* value equal to *acurve*.

3) if the new edge splits a face, then:

A) let *F* be an INTEGER value equal to the *FACE_ID* of the face being split.

B) delete from *ST_TOPO_GEO.ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F*.

C) generate two unique face id INTEGER values *newface1* and *newface2*.

D) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split.

E) let *MBR1* be the *ST_Polygon* value equal to the value of *ST_GetFaceGeometry(atopology,newface1).ST_Envelope()*.

F) let *MBR2* be the *ST_Polygon* value equal to the value of *ST_GetFaceGeometry(atopology,newface2).ST_Envelope()*.

G) insert into *ST_TOPO_GEO*.*ST_FACE* values (*atopology*, *newface1*, *MBR1*).

H) insert into *ST_TOPO_GEO*.*ST_FACE* values (*atopology*, *newface2*, *MBR2*).

4) return *anedgeid*.

x) To determine the two new face ID values, select *LEFT_FACE* and *RIGHT_FACE* from *ST_TOPO_GEO*.*ST_EDGE* where *EDGE_ID* is equal to *anedgeid*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.13 ST_AddEdgeModFace Function

Purpose

Add a new edge and, if in doing so it splits a face, modify the original face and add a new face.

Definition

```
CREATE FUNCTION ST_AddEdgeModFace
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddEdgeModFace(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anode*,
   c) an INTEGER value *anothernode*,
   d) an *ST_Curve* value *acurve*.
2) For the function *ST_AddEdgeModFace(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   g) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.
   h) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

Case:

i) If there is no row in *ST_TOPO_GEO.ST_NODE* with a *NODE_ID* value equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

ii) If there is no row in *ST_TOPO_GEO.ST_NODE* with a *NODE_ID* value equal to *anothernode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

iii) Let *P1* be the *ST_Point* value equal to *ST_TOPO_GEO.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to *acurve.ST_StartPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

iv) Let *P2* be the *ST_Point* value equal to *ST_TOPO_GEO.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

v) Let *N1* be a *NODE_ID* value in *ST_TOPO_GEO.ST_NODE* such that *N1* does not exist as either an *ST_TOPO_GEO.ST_EDGE.START_NODE* or *ST_TOPO_GEO.ST_EDGE.END_NODE* value. Let *G1* be the *ST_TOPO_GEO.ST_NODE.GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

vi) Let *G* be an *ST_TOPO_GEO.ST_EDGE.GEOMETRY* value. If *acurve.ST_Crosses(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses an edge*.

vii) If there exists a row in *ST_TOPO_GEO.ST_EDGE* with a *START_NODE* value equal to *anode* and an *END_NODE* value equal to *anothernode* and a *GEOMETRY* value equal to *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – coincident edge*.

viii) If there exists a row in *ST_TOPO_GEO.ST_EDGE* with a *START_NODE* value equal to *anothernode* and an *END_NODE* value equal to *anode* and a *GEOMETRY* value equal to *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – coincident edge*.

ix) Otherwise:

1) generate a unique edge id INTEGER value *anedgeid*.

2) insert into *ST_TOPO_GEO.ST_EDGE* a new row with:

A) a *TOPOLOGY* value equal to *atopology*.

B) an *EDGE_ID* value equal to *anedgeid*.

C) a *START_NODE* value equal to *anode*.

D) an *END_NODE* value equal to *anothernode*.

E) *NEXT_LEFT_EDGE*, *NEXT_RIGHT_EDGE*, *LEFT_FACE*, and *RIGHT_FACE* values as appropriate.

F) a *GEOMETRY* value equal to *acurve*.

3) if the new edge splits a face, then:

A) let *F* be an INTEGER value equal to the *FACE_ID* of the face being split.

B) update *ST_TOPO_GEO.ST_FACE* set MBR equal to the value of *ST_GetFaceGeometry(atopology,F1).ST_Envelope()* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F*.

C) generate a unique face id INTEGER value *newface*.

D) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split.

E) let *MBR* be the ST_Polygon value equal to the value of *ST_GetFaceGeometry(atopology,newface).ST_Envelope()*.

F) insert into *ST_TOPO_GEO.ST_FACE* values (*atopology*, *newface*, *MBR*).

4) return *anedgeid*.

x) To determine the ID values for the new face and the modified face, select *LEFT_FACE* and *RIGHT_FACE* from *ST_TOPO_GEO*.*ST_EDGE* where *EDGE_ID* is equal to *anedgeid*.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.14 ST_RemEdgeNewFace Function

Purpose

Remove an edge and, if the removed edge separated two faces, delete the original faces and replace them with one new face.

Definition

```
CREATE FUNCTION ST_RemEdgeNewFace
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_RemEdgeNewFace(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*.
2) For the function *ST_RemEdgeNewFace(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
      ii) Otherwise:
         1) delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*.
         2) if the edge removal results in the healing of two faces, then let *F1* and *F2* be the INTEGER values equal to the *FACE_ID*s of the faces to be healed.

Case:

A) If *F1* is equal to 0 (zero) then delete from *ST_TOPO_GEO.ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F2*.

B) If *F2* is equal to 0 (zero) then delete from *ST_TOPO_GEO.ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F1*.

C) Otherwise:

i) delete from *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F1*.

ii) delete from *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F2*.

iii) generate a unique face id INTEGER value *newface*.

iv) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed.

v) let *MBR* be the *ST_Polygon* value equal to the value of *ST_GetFaceGeometry(atopology,newface).ST_Envelope()*.

vi) insert into *ST_TOPO_GEO*.*ST_FACE* values (*atoploogy*, *newface*, *MBR*).

vii) return *newface*.

iii) The nodes in ST_TOPO_GEO.ST_NODE having a NODE_ID value equal to ST_TOPO_GEO.ST_EDGE.START_NODE and ST_TOPO_GEO.ST_EDGE.END_NODE where EDGE_ID is equal to anedge are not deleted.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.15 ST_RemEdgeModFace Procedure

Purpose

Remove an edge and, if the removed edge separated two faces, heal the two faces by modifying one of the faces and delete the other.

Definition

```
CREATE PROCEDURE ST_RemEdgeModFace
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_RemEdgeModFace(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *anedge*.
2) For the procedure *ST_RemEdgeModFace(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:
      i) If there is no row in *ST_TOPO_GEO*.*ST_EDGE* with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
      ii) Otherwise:
         1) delete from *ST_TOPO_GEO*.*ST_EDGE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge*.
         2) if the edge removal results in the healing of two faces, then let *F1* and *F2* be the INTEGER values equal to the *FACE_ID*s of the faces to be healed.

            Case:

A) If *F1* is equal to 0 (zero) then delete from *ST_TOPO_GEO.ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F2*.

B) If *F2* is equal to 0 (zero) then delete from *ST_TOPO_GEO.ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F1*.

C) Otherwise:

i) delete from *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F2*.

ii) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed.

iii) update *ST_TOPO_GEO*.*ST_FACE* set MBR equal to the value of *ST_GetFaceGeometry(atopology,F1).ST_Envelope()* where *TOPOLOGY* is equal to *atopology* and *FACE_ID* is equal to *F1*.

iii) The nodes in *ST_TOPO_GEO*.*ST_NODE* having a *NODE_ID* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* and *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* where *TOPOLOGY* is equal to *atopology* and *EDGE_ID* is equal to *anedge* are not deleted.

iv) If neither *F1* nor *F2* is equal to 0 (zero), then the choice of which face to modify and which to delete is implementation-dependent.

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.16 ST_GetFaceGeometry Function

Purpose

Return the exact geometry of a face.

Definition

```
CREATE FUNCTION ST_GetFaceGeometry
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    aface INTEGER)
   RETURNS ST_Surface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_GetFaceGeometry(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*,
   b) an INTEGER value *aface*.
2) For the function *ST_GetFaceGeometry(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) If *aface* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:

      i) If *aface* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – universal face has no geometry*.
      ii) If there is no row in *ST_TOPO_GEO*.*ST_FACE* with a *FACE_ID* value equal to *aface*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent face*.
      iii) Otherwise:
         1) let *T* be the table returned by *ST_GetFaceEdges(atopology, aface)*.

2) for each row in *T*, let *G* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to the value of *edge* from that row. The *ST_Curve* value shall have the same spatial reference system identifier as the *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* value.

3) let *G′* be the *ST_Curve* value derived from *G*, corrected for the direction of *edge*; for each value of *G*:

   Case:

   A) if *edge* is greater than 0 (zero), then *G′* is equal to *G*.

   B) otherwise, construct a value for *G′* by reversing the direction of *G*.

4) case:

   A) if all *G′ ST_Curve* values have the same spatial reference system identifier and a valid *ST_Surface* geometry value *geometry* can be constructed from the values of *G′*, connecting them in the order specified by the *sequence* values in *T*, then

      I) create the valid *geometry* value accordingly, with the common spatial reference system identifier.

      II) return *geometry*.

   B) otherwise, return an empty set of type *ST_Surface*.

5) return *geometry*.

11.3.17 ST_InitTopoGeo Procedure

Purpose

Create schema and views for a Topology-geometry.

Definition

```
CREATE PROCEDURE ST_InitTopoGeo
   (atopology CHARACTER VARYING(ST_MaxTopologyName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_InitTopoGeo(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *atopology*.

2) For the procedure *ST_InitTopoGeo(CHARACTER VARYING)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*atopology*' returns a value equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – schema already exists*.

   c) Otherwise:

      i) create a schema with a name equal to *atopology*.

      ii) for schema *atopology*, create ST_NODE, ST_EDGE, and ST_FACE views in accordance with Subclause 10.1.2, "ST_NODE view", Subclause 10.1.3, "ST_EDGE view", and Subclause 10.1.4, "ST_FACE view".

      iii) insert into *ST_TOPO_GEO*.*ST_FACE* values (*atopology*, 0, NULL).

11.3.18 ST_CreateTopoGeo Procedure

Purpose

Create a topologically consistent Topology-geometry from a collection of geometry values.

Definition

```
CREATE PROCEDURE ST_CreateTopoGeo
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    ageomcollection ST_GeomCollection)
   LANGUAGE SQL
   NOT DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_CreateTopoGeo(CHARACTER VARYING, ST_GeomCollection)* takes the following input parameters:

   a) a CHARACTER VARYING value *atopology*,

   b) an *ST_GeomCollection* value *ageomcollection*.

2) For the procedure *ST_CreateTopoGeo(CHARACTER VARYING, ST_GeomCollection)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.

   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.

   d) If *ageomcollection* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   e) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*atopology*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   f) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_NODE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   g) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_EDGE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   h) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_FACE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   i) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

Case:

i) If SELECT COUNT(*) FROM *ST_TOPO_GEO.ST_NODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

ii) If SELECT COUNT(*) FROM *ST_TOPO_GEO.ST_EDGE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

iii) If SELECT COUNT(*) FROM *ST_TOPO_GEO.ST_FACE* returns a value greater than 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

iv) If SELECT FACE_ID FROM *ST_TOPO_GEO.ST_FACE* returns a value other than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – invalid universal face.*

v) Otherwise:

1) using the geometry values in *ageomcollection*, create the corresponding consistent topology. This may require that the geometry collection first be modified to satisfy the requirements for topological consistency (4.3, “Topology-Geometry”), such as splitting *ST_Curves* where they cross and adding *ST_Point* values at their ends.

2) for each face:

A) let *faceid* be the generated unique face id positive INTEGER value not equal to 0 (zero).

B) let *mbr* be the *ST_Polygon* value which represents the minimum bounding rectangle of the face.

C) insert into *ST_TOPO_GEO.ST_FACE* values (*atopology*, *faceid*, *mbr*).

3) for each node:

A) let *nodeid* be the generated unique node id positive INTEGER value.

B) let *geometry* be the *ST_Point* value which specifies the node location.

C) case:

I) if the node is an isolated node, then let *face* be the INTEGER value equal to the face id of the containing face.

II) otherwise, let *face* be the null value.

D) insert into *ST_TOPO_GEO.ST_NODE* values (*atopology*, *nodeid*, *geometry*, *face*).

4) for each edge:

A) let *edgeid* be the generated unique available edge id positive INTEGER value.

B) let *startnode* be the INTEGER value equal to the node id of the node at the start of the edge.

C) let *endnode* be the INTEGER value equal to the node id of the node at the end of the edge.

D) let *nextleft* be the INTEGER value equal to the edge id of the next edge of the face on the left (when looking in the direction from its start node to its end node), moving counterclockwise around the face boundary.

E) let *ENL* be the INTEGER value equal to the node id of the end node of the edge with edge id equal to *nextleft*. If *endnode* is equal to *ENL* then set *nextleft* = - (*nextleft*).

F) let *nextright* be the INTEGER value equal to the edge id of the next edge of the face on the right (when looking in the direction from the start node edge to its end node), moving counterclockwise around the face boundary.

G) let *leftface* be the INTEGER value equal to the face id of the face on the left side of the edge (when looking in the direction from the start node of the edge to its end node).

H) let *rightface* be the INTEGER value equal to the face id of the face on the right side of the edge (when looking in the direction from the start node of the edge to its end node).

I) let *geometry* be the *ST_Curve* value which represents the geometry of the edge, in the same direction as the edge.

J) insert into *ST_TOPO_GEO*.*ST_EDGE* values (*atopology*, *edgeid*, *startnode*, *endnode*, *nextleft*, *nextright*, *leftface*, *rightface*, *geometry*).

3) All geometry values in the *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* base tables in rows which have the same *TOPOLOGY* column value shall have the same spatial reference system identifier.

11.3.19 ST_ValidateTopoGeo Function

Purpose

Return a table containing possible topological inconsistencies for a Topology-geometry.

Definition

```
CREATE FUNCTION ST_ValidateTopoGeo
   (atopology CHARACTER VARYING(ST_MaxTopologyName))
   RETURNS TABLE
      (error CHARACTER VARYING(30),
       primitive1 INTEGER,
       primitive2 INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The function *ST_ValidateTopoGeo(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology*.
2) For the function *ST_ValidateTopoGeo(CHARACTER VARYING)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.
   c) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   d) Otherwise, for rows in *ST_TOPO_GEO*.*ST_NODE*, *ST_TOPO_GEO*.*ST_EDGE*, and *ST_TOPO_GEO*.*ST_FACE* where *TOPOLOGY* is equal to *atopology*:

      Case:

      i) If SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_NODE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_EDGE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_FACE* returns a value less than or equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – empty topology*.
      ii) Otherwise,
         1) let *T* be a table value consisting of the following three columns:
            A) a column *error* of type CHARACTER VARYING, which identifies the type of inconsistency found in *atopology*.
            B) a column *primitive1* of type INTEGER, which contains the node, edge, or face id of the first offending topology primitive.

C) a column *primitive2* of type INTEGER, which contains the node, edge, or face id of the second offending topology primitive.

2) for each pair of coincident nodes:

A) let *E* be the CHARACTER VARYING value equal to 'coincident nodes'.

B) let *N1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*NODE_ID* for a node in *atopology*.

C) let *G1* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N1*.

D) let *N2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*NODE_ID* for another node in *atopology*.

E) let *G2* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.

F) for all values of *N1* and *N2* such that *G1* is equal to *G2*, insert into *T* values (*E*, *N1*, *N2*).

3) for each node crossed by an edge:

A) let *E* be the CHARACTER VARYING value equal to 'edge crossed node'.

B) let *N1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*NODE_ID* for a node in *atopology*.

C) let *G1* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N1*.

D) let *E2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*.

E) let *G2* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E2*.

F) for all values of *N1* and *E2* such that the value returned by *G2.ST_Crosses(G1)* is equal to 1 (one), insert into *T* values (*E*, *N1*, *E2*).

4) for each non-simple edge:

A) let *E* be the CHARACTER VARYING value equal to 'edge not simple'.

B) let *E1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*.

C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E1*.

D) let *E2* be equal to the null value.

E) for all values of *E1* such that the value returned by *G1.ST_IsSimple()* is equal to 0 (zero), insert into *T* values (*E*, *E1*, *E2*).

F) let *ENR* be the INTEGER value equal to the node id of the end node of the edge with edge id equal to *nextright*. If *startnode* is equal to *ENR* then set *nextright* = -(*nextright*).

5) for each edge crossing another edge:

A) let *E* be the CHARACTER VARYING value equal to 'edge crosses edge'.

B) let *E1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*.

C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E1*.

D) let *E2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for another edge in *atopology*.

E) let *G2* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E2*.

F) for all values of *E1* and *E2* such that the value returned by *G1.ST_Crosses(G2)* is equal to 1 (one), insert into *T* values (*E*, *E1*, *E2*).

6) for each edge having a geometry with a start point not equal to the geometry of its start node:

A) let *E* be the CHARACTER VARYING value equal to 'geometry mismatch'.

B) let *E1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*.

C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E1*.

D) let *N2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*START_NODE* where *EDGE_ID* is equal to *E1*.

E) let *G2* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.

F) for all values of *E1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_StartPoint())* is not equal to 1 (one), insert into *T* values (*E*, *E1*, *N2*).

7) for each edge having a geometry with an end point not equal to the geometry of its end node:

A) let *E* be the CHARACTER VARYING value equal to 'geometry mismatch'.

B) let *E1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*.

C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E1*.

D) let *N2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*END_NODE* where *EDGE_ID* is equal to *E1*.

E) let *G2* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.

F) for all values of *E1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_EndPoint())* is not equal to 1 (one), insert into *T* values (*E*, *E1*, *N2*).

8) for each face with no edges:

A) let *E* be the CHARACTER VARYING value equal to 'face without edges'.

B) let *F1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for a face in *atopology*.

C) let *F2* be equal to the null value.

D) for all values of *F1* such that the value returned by SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_EDGE* WHERE *LEFT_FACE = F1* is equal to 0 (zero) and SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_EDGE* WHERE *RIGHT_FACE = F1* is also equal to 0 (zero), insert into *T* values (*E*, *F1*, *F2*).

9) for each face overlapping another face:

A) let *E* be the CHARACTER VARYING value equal to 'face overlaps face'.

B) let *F1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for a face in *atopology*.

C) let G1 be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F1)*.

D) let *F2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for another face in *atopology*.

E) let G2 be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F2)*.

F) for all values of *F1* and *F2* such that the value returned by *G1.ST_Overlaps(G2)* is equal to 1 (one), insert into *T* values (*E*, *F1*, *F2*).

10) for each face contained within another face:

A) let *E* be the CHARACTER VARYING value equal to 'face within face'.

B) let *F1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for a face in *atopology*.

C) let G1 be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F1)*.

D) let *F2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for another face in *atopology*.

E) let G2 be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F2)*.

F) for all values of *F1* and *F2* such that the value returned by *G1.ST_Within(G2)* is equal to 1 (one), insert into *T* values (*E*, *F1*, *F2*).

11) missing universal face:

A) let *E* be the CHARACTER VARYING value equal to 'no universal face'.

B) if SELECT COUNT(*) FROM *ST_TOPO_GEO*.*ST_FACE* WHERE *FACE_ID* = 0 (zero) is equal to 0 (zero), insert into *T* values (*E*, NULL, NULL).

12) for each non-universal face for which a valid geometry cannot be created:

A) let *E* be the CHARACTER VARYING value equal to 'invalid face geometry'.

B) let *F1* be the non-zero INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for a non-universal face in *atopology*.

C) let *G1* be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology, F1)*.

D) for all values of *F1* such that the value returned by *G1.ST_IsEmpty()* is equal to 1 (one), insert into *T* values (*E*, *F1*, NULL).

13) if all geometries do not have the same spatial reference system identifier:

A) let *E* be the CHARACTER VARYING value equal to 'mixed SRIDs'.

B) let *N1* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*NODE_ID* for a node in *atopology*.

C) let *G1* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N1*.

D) let *S1* be the INTEGER value returned by *G1.ST_SRID()*.

E) let *N2* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_NODE*.*NODE_ID* for another node in *atopology*.

F) let *G2* be the *ST_Point* value equal to *ST_TOPO_GEO*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.

G) let *S2* be the INTEGER value returned by *G2.ST_SRID()*.

H) let *E3* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_EDGE*.*EDGE_ID* for an edge in *atopology*.

I) let *G3* be the *ST_Curve* value equal to *ST_TOPO_GEO*.*ST_EDGE*.*GEOMETRY* where *EDGE_ID* is equal to *E3*.

J) let *S3* be the INTEGER value returned by *G3.ST_SRID()*.

K) let *F4* be the INTEGER value equal to *ST_TOPO_GEO*.*ST_FACE*.*FACE_ID* for a face in *atopology*.

L) let *G4* be the *ST_Surface* value equal to *ST_TOPO_GEO*.*ST_FACE*.MBR where *FACE_ID* is equal to *F4*.

M) let *S4* be the INTEGER value returned by *G4.ST_SRID()*.

N) if any value of *N2*, any value of *E3* or any value of *F4* having a corresponding non-null value for G4 has a corresponding value of *S2*, *S3* or *S4*, respectively, such that *S2* is not equal to *S1* or *S3* is not equal to *S1* or *S4* is not equal to *S1*, insert into *T* values (*E*, NULL, NULL).

12 Topology-Network

12.1 Topo-Net Network Schema

12.1.1 Introduction

The Topo-Net Network Schema views are defined as being in a schema named <network-name> enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views may be granted to individual users so that they can be queried. To update a Topo-Net, a user shall be granted SELECT privlilege on the views for the Topo-Net and EXECUTE privilege on the Topo-Net routines provided. Roles can be used to enable update on a selective set of Topo-Nets. These views are updated only by the topology functions provided.

An implementation may define objects that are associated with <network-name> that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

All of the topological primitives contained in the views owned by the <network-name> schema constitute a single, topologically consistent, topology. Other topologies (e.g., covering a different spatial extent, existing at a different level of abstraction, or associated with a different set of features) can exist in another, independent <network-name> schema.

12.1.2 ST_NODE view

Purpose

Contains the node type of topological primitives (ST_Node) contained in the <network-name> Topo-Net.

Definition

```
CREATE VIEW ST_NODE AS
   SELECT NODE_ID, GEOMETRY
      FROM ST_TOPO_NET.ST_NODE
      WHERE NETWORK = '<network-name>'
```

12.1.3 ST_LINK view

Purpose

Contains the link type of topological primitives (ST_Link) contained in the <network-name> Topo-Net.

Definition

```
CREATE VIEW ST_LINK AS
   SELECT LINK_ID, START_NODE, END_NODE, GEOMETRY
      FROM ST_TOPO_NET.ST_LINK
      WHERE NETWORK = '<network-name>'
```

12.2 Topo-Net Definition Schema

12.2.1 Introduction

The only purpose of this Topo-Net Definition Schema is to provide a data model to support Topo-Net views and to assist understanding. The base tables of this Topo-Net Definition Schema are defined as being in a schema named *ST_TOPO_NET*. The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1) The function of the definition is stated.
2) The SQL definition of the object is presented as a <table definition>.
3) An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

12.2.2 ST_NODE base table

Purpose

Contains the node type of topological primitives (ST_Node) contained in topology-networks.

Definition

```
CREATE TABLE ST_NODE
   (
   NETWORK CHARACTER VARYING(ST_MaxNetworkName),
   NODE_ID INTEGER NOT NULL,
   GEOMETRY ST_Point,

   CONSTRAINT ST_NODE_PRIMARY_KEY PRIMARY KEY(NETWORK, NODE_ID)
   )
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The value of *NETWORK* is the network name which distinguishes which nodes are in the network.
2) Let *N* be the value of *NETWORK* for a given node.
3) The value of *NODE_ID* is the identifier of the node unique among all nodes with a *NETWORK* value equal to *N*.
4) The value of *GEOMETRY* is the spatial location of the node. For logical networks, *GEOMETRY* is the null value.
5) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.2.3 ST_LINK base table

Purpose

Contains the link type of topological primitives (ST_Link) contained in topology-networks.

Definition

```
CREATE TABLE ST_LINK
   (
   NETWORK CHARACTER VARYING(ST_MaxNetworkName),
   LINK_ID INTEGER NOT NULL,
   START_NODE INTEGER NOT NULL,
   END_NODE INTEGER NOT NULL,
   GEOMETRY ST_Curve,

   CONSTRAINT ST_LINK_PRIMARY_KEY PRIMARY KEY(NETWORK, LINK_ID),
   CONSTRAINT START_NODE_EXISTS FOREIGN KEY(NETWORK, START_NODE)
         REFERENCES ST_NODE(NETWORK, NODE_ID),
   CONSTRAINT END_NODE_EXISTS FOREIGN KEY(NETWORK, END_NODE)
         REFERENCES ST_NODE(NETWORK, NODE_ID)
   )
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The value of *NETWORK* is the network name which distinguishes which nodes are in the network.
2) Let *N* be the value of *NETWORK* for a given link.
3) The value of *LINK_ID* is the identifier of the link unique among all links with a *NETWORK* value equal to *N*.
4) The value of *START_NODE* is the unique identifier of the node at the start of the link; the start node shall have a value of *NETWORK* equal to *N*.
5) The value of *END_NODE* is the unique identifier of the node at the end of the link; the end node shall have a value of *NETWORK* equal to *N*.
6) The value of *GEOMETRY* is the geometry of the link. For logical networks, *GEOMETRY* is the null value.
7) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3 Topo-Net Routines

12.3.1 ST_AddIsoNetNode Function

Purpose

Insert a row into the <network-name>.*ST_NODE* view for an isolated node.

Definition

```
CREATE FUNCTION ST_AddIsoNetNode
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_AddIsoNetNode(CHARACTER VARYING, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an *ST_Point* value *apoint*.
2) For the function *ST_AddIsoNetNode(CHARACTER VARYING, ST_Point)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) Otherwise:
      i) generate a unique node id INTEGER value *anodeid*.
      ii) insert into *ST_TOPO_NET*.*ST_NODE* values (*anetwork*, *anodeid*, *apoint*).
      iii) return *anodeid*.
3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.2 ST_MoveIsoNetNode Procedure

Purpose

Update the <network-name>.*ST_NODE*.*GEOMETRY* value of an isolated node.

Definition

```
CREATE PROCEDURE ST_MoveIsoNetNode
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    anode INTEGER,
    apoint ST_Point)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_MoveIsoNetNode(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *anode*,
   c) an *ST_Point* value *apoint*.
2) For the procedure *ST_MoveIsoNetNode(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If any *ST_TOPO_NET*.*ST_LINK*.*START_NODE* or *ST_TOPO_NET*.*ST_LINK*.*END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.
      ii) Otherwise, update *ST_TOPO_NET*.*ST_NODE*, set the *GEOMETRY* value equal to *apoint* where *NETWORK* is equal to *anetwork* and *NODE_ID* is equal to *anode*.
3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.3 ST_RemIsoNetNode Procedure

Purpose

Delete a row in the <network-name>.*ST_NODE* view corresponding to an isolated node.

Definition

```
CREATE PROCEDURE ST_RemIsoNetNode
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    anode INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_RemIsoNetNode(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *anode*.
2) For the procedure *ST_RemIsoNetNode(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If any *ST_TOPO_NET*.*ST_LINK*.*START_NODE* or *ST_TOPO_NET*.*ST_LINK*.*END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.
      ii) Otherwise, delete from *ST_TOPO_NET*.*ST_NODE* where *NETWORK* is equal to *anetwork* and *NODE_ID* is equal to *anode*.

12.3.4 ST_AddLink Function

Purpose

Insert a row for a link into the <network-name>.*ST_LINK* view connecting two existing nodes.

Definition

```
CREATE FUNCTION ST_AddLink
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_AddLink(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *anode*,
   c) an INTEGER value *anothernode*,
   d) an *ST_Curve* value *acurve*.
2) For the function *ST_AddLink(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* WHERE *NODE_ID* is equal to *anode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node.*

ii) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* WHERE *NODE_ID* is equal to *anothernode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node.*

iii) If *acurve* is not the null value, then:

1) let *P1* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to the null value and *P1* is not equal to *acurve.ST_StartPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

2) let *P2* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to the null value and *P2* is not equal to *acurve.ST_EndPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

iii) Otherwise:

1) generate a unique link id INTEGER value *alinkid*.

2) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *alinkid*, *anode*, *anothernode*, *acurve*).

3) return *alinkid*.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.5 ST_ChangeLinkGeom Procedure

Purpose

Update the <network-name>.*ST_LINK*.*GEOMETRY* value.

Definition

```
CREATE PROCEDURE ST_ChangeLinkGeom
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    acurve ST_Curve)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_ChangeLinkGeom(CHARACTER VARYING, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*,
   c) an *ST_Curve* value *acurve*.
2) For the procedure *ST_ChangeLinkGeom(CHARACTER VARYING, INTEGER, ST_Curve)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If *acurve* is not the null value, then:
         1) let *P1* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to the value of *ST_TOPO_NET*.*ST_LINK*.*START_NODE* where *LINK_ID* is equal to *alink*.

            Case;
            A) If *P1* is equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – null node geometry*.

B) Otherwise, if *P1* is not equal to *acurve.ST_StartPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

2) let *P2* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to the value of *ST_TOPO_NET*.*ST_LINK*.*END_NODE* where *LINK_ID* is equal to *anothernode*.

Case:

A) if *P2* is equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – null node geometry*.

B) otherwise, if *P2* is not equal to *acurve.ST_EndPoint()*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

ii) Otherwise, update *ST_TOPO_NET*.*ST_LINK*, set the *GEOMETRY* value equal to *acurve where NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.6 ST_RemoveLink Procedure

Purpose

Delete a row in the <network-name>.*ST_LINK* view corresponding to a link.

Definition

```
CREATE PROCEDURE ST_RemoveLink
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_RemoveLink(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*.
2) For the procedure *ST_RemoveLink(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:
      i) delete from *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*.

12.3.7 ST_InitTopoNet Procedure

Purpose

Create schema and views for a topology-network.

Definition

```
CREATE PROCEDURE ST_InitTopoNet
   (anetwork CHARACTER VARYING(ST_MaxNetworkName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_InitTopoNet(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*.
2) For the procedure *ST_InitTopoNet(CHARACTER VARYING)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*anetwork*' returns a value equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – schema already exists*.
   c) Otherwise:
      i) create a schema with a name equal to *anetwork*.
      ii) for schema *anetwork*, create ST_NODE and ST_LINK views in accordance with Subclause 11.1.2, "ST_NODE view" and Subclause 11.1.3 "ST_LINK view".

12.3.8 ST_NewLogLinkSplit Function

Purpose

Split a link in a logical network by creating a new node along an existing link, deleting the original link and replacing it with two new links.

Definition

```
CREATE FUNCTION ST_NewLogLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function ST_NewLogLinkSplit(CHARACTER VARYING, INTEGER) takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*.
2) For the function *ST_NewLogLinkSplit(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
      ii) If the value of *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *alink* is not equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – not a logical link*.
      iii) Otherwise:
         1) generate a unique node id INTEGER value *newnode*.

2) insert into *ST_TOPO_NET*.*ST_NODE* values (*anetwork*, *newnode*, NULL).

3) select from *ST_TOPO_NET*.*ST_LINK* *START_NODE* and *ST_TOPO_NET*.*END_NODE* into *oldstart* and *oldend* where *LINK_ID* is equal to *alink*.

4) delete from *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*.

5) generate two unique link id INTEGER values *newlink1* and *newlink2*.

6) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink1*, *oldstart*, *newnode*, NULL).

7) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink2*, *newnode*, *oldend*, NULL).

8) return *newnode*.

iv) Both new links have the same direction as the link being split.

v) To determine the two new link ID values, select *LINK_ID* from *ST_TOPO_NET*.*ST_LINK* where *START_NODE* or *END_NODE* is equal to *newnode*.

12.3.9 ST_ModLogLinkSplit Function

Purpose

Split a link in a logical network by creating a new node along an existing link, modifying the original link and adding a new link.

Definition

```
CREATE FUNCTION ST_ModLogLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_ModLogLinkSplit(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*.
2) For the function *ST_ModLogLinkSplit(CHARACTER VARYING, INTEGER)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
      ii) If the value of *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *alink* is not equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – not a logical link*.
      iii) Otherwise:
         1) generate a unique node id INTEGER value *newnode*.

2) insert into *ST_TOPO_NET*.*ST_NODE* values (*anetwork*, *newnode*, NULL).

3) select from *ST_TOPO_NET*.*ST_LINK END_NODE* into *oldend* where *LINK_ID* is equal to *alink*.

4) generate a unique link id INTEGER value *newlink*.

5) update *ST_TOPO_NET*.*ST_LINK* set the *END_NODE* value equal to *newnode* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*.

6) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink*, *newnode*, *oldend*, NULL).

7) return *newnode*.

iv) The new and modified links have the same direction as the link being split.

v) To determine the new link ID value, select *LINK_ID* from *ST_TOPO_NET*.*ST_LINK* where *START_NODE* is equal to *newnode*.

12.3.10 ST_NewGeoLinkSplit Function

Purpose

Split a link in a network with geometry by creating a new node along an existing link, deleting the original link and replacing it with two new links.

Definition

```
CREATE FUNCTION ST_NewGeoLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_NewGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*,
   c) an *ST_Point* value *apoint*.
2) For the function *ST_NewGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.

ii) Let *G* be equal to the *ST_Curve* value of *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *alink*. If *G* is equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – link has null geometry*.

iii) If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on link*.

iv) Otherwise:

1) generate a unique node id INTEGER value *newnode*.

2) insert into *ST_TOPO_NET*.*ST_NODE* values (*anetwork*, *newnode*, *apoint*).

3) select from *ST_TOPO_NET*.*ST_LINK* *START_NODE*, *END_NODE*, and *GEOMETRY* into *oldstart*, *oldend*, and *oldgeom* where *LINK_ID* is equal to *alink*.

4) delete from *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*.

5) generate two unique link id INTEGER values *newlink1* and *newlink2*.

6) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*.

7) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink1*, *oldstart*, *newnode*, *newgeom1*).

8) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink2*, *newnode*, *oldend*, *newgeom2*).

9) return *newnode*.

v) Both new links have the same direction as the link being split.

vi) To determine the two new link ID values, select *LINK_ID* from *ST_TOPO_NET*.*ST_LINK* where *START_NODE* or *END_NODE* is equal to *newnode*.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.11 ST_ModGeoLinkSplit Function

Purpose

Split a link in a network with geometry by creating a new node along an existing link, modifying the original link and adding a new link.

Definition

```
CREATE FUNCTION ST_ModGeoLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_ModGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*,
   c) an *ST_Point* value *apoint*.
2) For the function *ST_ModGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:

      i) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.

ii) Let *G* be equal to the *ST_Curve* value of *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *alink*. If *G* is equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – link has null geometry*.

iii) If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on link*.

iv) Otherwise:

1) generate a unique node id INTEGER value *newnode*.

2) insert into *ST_TOPO_NET*.*ST_NODE* values (*anetwork*, *newnode*, *apoint*).

3) select from *ST_TOPO_NET*.*ST_LINK* *END_NODE* and *GEOMETRY* into *oldend* and *oldgeom* where *LINK_ID* is equal to *alink*.

4) generate a unique link id INTEGER value *newlink*.

5) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*.

6) update *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*:

A) set the *END_NODE* value equal to *newnode*.

B) set the *GEOMETRY* value equal to *newgeom1*.

7) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink*, *newnode*, *oldend*, *newgeom2*).

8) return *newnode*.

v) The new and modified links have the same direction as the link being split.

vi) To determine the new link ID value, select *LINK_ID* from *ST_TOPO_NET*.*ST_LINK* where *START_NODE* is equal to *newnode*.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.12 ST_NewLinkHeal Function

Purpose

Heal two links by deleting the node connecting them, deleting both links, and replacing them with a new link whose direction is the same as the first link provided.

Definition

```
CREATE FUNCTION ST_NewLinkHeal
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    anotherlink INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_NewLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*,
   c) an INTEGER value *anotherlink*.
2) For the function *ST_NewLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anotherlink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
      ii) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.

iii) Check if links are connected:

1) let *COMMONNODE* be an INTEGER value.

2) let *CASE* be an INTEGER value.

3) let *S1* and *E1* be INTEGER values equal to the *ST_TOPO_NET*.*ST_LINK*.*START_NODE* and *ST_TOPO_NET*.*END_NODE* values, respectively, where *LINK_ID* is equal to *alink*.

4) let *S2* and *E2* be INTEGER values equal to the *ST_TOPO_NET*.*ST_LINK*.*START_NODE* and *ST_TOPO_NET*.*END_NODE* values, respectively, where *LINK_ID* is equal to *anotherlink*.

5) Case:

A) if *E1* is equal to *S2*, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to *E1*.

B) if *E1* is equal to *E2*, then set *CASE* equal to 2 and *COMMONNODE* equal to *E1*.

C) if *S1* is equal to *S2*, then set *CASE* equal to 3 and *COMMONNODE* equal to *S1*.

D) if *S1* is equal to *E2*, then set *CASE* equal to 4 and *COMMONNODE* equal to *S1*.

E) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected links*.

iv) If *COMMONNODE* is equal to the *ST_TOPO_NET*.*ST_LINK*.*START_NODE* or *ST_TOPO_NET*.*ST_LINK*.*END_NODE* value for any link other than the link whose *LINK_ID* is equal to either *alink* or *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – other links connected*.

v) Otherwise:

1) delete from *ST_TOPO_NET*.*ST_NODE* where *NETWORK* is equal to *anetwork* and *NODE_ID* is equal to *COMMONNODE*.

2) select from *ST_TOPO_NET*.*ST_LINK START_NODE*, *END_NODE*, and *GEOMETRY* into *oldstart1*, *oldend1*, and *oldgeom1* where *LINK_ID* is equal to *alink*.

3) select from *ST_TOPO_NET*.*ST_LINK START_NODE*, *END_NODE*, and *GEOMETRY* into *oldstart2*, *oldend2*, and *oldgeom2* where *LINK_ID* is equal to *anotherlink*.

4) delete from *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*.

5) delete from *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *anotherlink*.

6) generate a unique link id INTEGER value *newlink*.

7) case:

A) if *CASE* is equal to 1 (one) then:

I) Case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*.

II) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink*, *oldstart1*, *oldend2*, *newgeom*).

B) if *CASE* is equal to 2 then:

I) Case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*.

II) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink*, *oldstart1*, *oldstart2*, *newgeom*).

C) if *CASE* is equal to 3 then:

I) case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end.

II) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink*, *oldend2*, *oldend1*, *newgeom*).

D) if *CASE* is equal to 4 then:

I) case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*.

II) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *newlink*, *oldstart2*, *oldend1*, *newgeom*).

8) return *newlink*.

vi) The direction of the new link shall be the same as the direction of the first link supplied.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.13 ST_ModLinkHeal Procedure

Purpose

Heal two links by deleting the node connecting them, modifying the first link provided, and deleting the second link.

Definition

```
CREATE PROCEDURE ST_ModLinkHeal
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    anotherlink INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_ModLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an INTEGER value *alink*,
   c) an INTEGER value *anotherlink*.
2) For the procedure *ST_ModLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:
   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *anotherlink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   f) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:
      i) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
      ii) If there is no row in *ST_TOPO_NET*.*ST_LINK* with an *LINK_ID* value equal to *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
      iii) Check if links are connected:

1) let *COMMONNODE* be an INTEGER value.

2) let *CASE* be an INTEGER value.

3) let *S1* and *E1* be INTEGER values equal to the *ST_TOPO_NET*.*ST_LINK*.*START_NODE* and *ST_TOPO_NET*.*END_NODE* values, respectively, where *LINK_ID* is equal to *alink*.

4) let *S2* and *E2* be INTEGER values equal to the *ST_TOPO_NET*.*ST_LINK*.*START_NODE* and *ST_TOPO_NET*.*END_NODE* values, respectively, where *LINK_ID* is equal to *anotherlink*.

5) case:

A) if *E1* is equal to *S2*, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to *E1*.

B) if *E1* is equal to *E2*, then set *CASE* equal to 2 and *COMMONNODE* equal to *E1*.

C) if *S1* is equal to *S2*, then set *CASE* equal to 3 and *COMMONNODE* equal to *S1*.

D) if *S1* is equal to *E2*, then set *CASE* equal to 4 and *COMMONNODE* equal to *S1*.

E) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected links*.

iv) If *COMMONNODE* is equal to the *ST_TOPO_NET*.*ST_LINK*.*START_NODE* or *ST_TOPO_NET*.*ST_LINK*.*END_NODE* value for any link other than the link whose *LINK_ID* is equal to either *alink* or *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – other links connected*.

v) Otherwise:

1) delete from *ST_TOPO_NET*.*ST_NODE* where *NETWORK* is equal to *anetwork* and *NODE_ID* is equal to *COMMONNODE*.

2) select from *ST_TOPO_NET*.*ST_LINK GEOMETRY* into *oldgeom1* where *LINK_ID* is equal to *alink*.

3) select from *ST_TOPO_NET*.*ST_LINK START_NODE*, *END_NODE*, and *GEOMETRY* into *oldstart2*, *oldend2*, and *oldgeom2* where *LINK_ID* is equal to *anotherlink*.

4) delete from *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *anotherlink*.

5) case:

A) if *CASE* is equal to 1 (one) then:

I) case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*.

II) update *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*:

1) set the *END_NODE* value equal to *oldend2*.

2) set the *GEOMETRY* value equal to *newgeom*.

B) if *CASE* is equal to 2 then:

I) case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*.

II) update *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*:

1) set the *END_NODE* value equal to *oldstart2*.

2) set the *GEOMETRY* value equal to *newgeom*.

C) if *CASE* is equal to 3 then:

I) case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end.

II) update *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*:

1) set the *START_NODE* value equal to *oldend2*.

2) set the *GEOMETRY* value equal to *newgeom*.

D) if *CASE* is equal to 4 then:

I) case:

1) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value.

2) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*.

II) update *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork* and *LINK_ID* is equal to *alink*:

1) set the *START_NODE* value equal to *oldstart2*.

2) set the *GEOMETRY* value equal to *newgeom*.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.14 ST_LogiNetFromTGeo Procedure

Purpose

Create a logical topology-network from a Topology-geometry.

Definition

```
CREATE PROCEDURE ST_LogiNetFromTGeo
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    atopology CHARACTER VARYING(ST_MaxTopologyName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

2) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_LogiNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *anetwork*,

   b) a CHARACTER VARYING value *atopology*.

2) For the procedure *ST_LogiNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.

   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.

   d) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   e) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topology name*.

   f) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.

   g) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*anetwork*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   h) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*atopology*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

i) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*anetwork*' AND TABLE_NAME = '*ST_NODE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

j) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*anetwork*' AND TABLE_NAME = '*ST_LINK*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

k) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_NODE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

l) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_EDGE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

m) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

   Case:

   i) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

   ii) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_LINK* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

   iii) Otherwise:

      1) for nodes in *atopology*, insert into *ST_TOPO_NET*.*ST_NODE* select *anetwork*, *NODE_ID*, NULL from *atopology*.*ST_NODE*.

      2) for edges in *atopology*, insert into *ST_TOPO_NET*.*ST_LINK* select *anetwork*, *EDGE_ID*, *START_NODE*, *END_NODE*, NULL from *atopology*.*ST_EDGE*.

12.3.15 ST_SpatNetFromTGeo Procedure

Purpose

Create a spatial topology-network from a Topology-geometry.

Definition

```
CREATE PROCEDURE ST_SpatNetFromTGeo
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    atopology CHARACTER VARYING(ST_MaxTopologyName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.
2) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_SpatNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) a CHARACTER VARYING value *atopology*.
2) For the procedure *ST_SpatNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If *atopology* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid topoloyg name*.
   f) If the user has not been granted SELECT privilege on *atopology*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – topology privilege denied*.
   g) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*anetwork*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.
   h) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*atopology*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

i) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*anetwork*' AND TABLE_NAME = '*ST_NODE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

j) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*anetwork*' AND TABLE_NAME = '*ST_LINK*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

k) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_NODE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

l) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*atopology*' AND TABLE_NAME = '*ST_EDGE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

m) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

Case:

i) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

ii) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_LINK* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

iii) Otherwise:

1) for nodes in *atopology*, insert into *ST_TOPO_NET*.*ST_NODE* select *anetwork*, *NODE_ID*, *GEOMETRY* from *ST_TOPO_GEO*.*ST_NODE*.

2) for edges in *atopology*, insert into *ST_TOPO_NET*.*ST_LINK* select *anetwork*, *EDGE_ID*, *START_NODE*, *END_NODE*, *GEOMETRY* from *ST_TOPO_GEO*.*ST_EDGE*.

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.16 ST_SpatNetFromGeom Procedure

Purpose

Create a topology-network from a collection of geometry values.

Definition

```
CREATE PROCEDURE ST_SpatNetFromGeom
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    ageomcollection ST_GeomCollection)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The procedure *ST_SpatNetFromGeom(CHARACTER VARYING, ST_GeomCollection)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*,
   b) an ST_GeomCollection value *ageomcollection*.
2) For the procedure *ST_SpatNetFromGeom(CHARACTER VARYING, ST_GeomCollection)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) If *ageomcollection* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '*anetwork*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.
   f) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*anetwork*' AND TABLE_NAME = '*ST_NODE*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.
   g) If SELECT COUNT(*) FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = '*anetwork*' AND TABLE_NAME = '*ST_LINK*' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.
   h) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:

      i) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

ii) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_LINK* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

iii) Otherwise:

1) using the geometry values in *ageomcollection*, create a corresponding set of network primitives (nodes and links).

2) for each node:

A) let *nodeid* be the generated unique node id INTEGER value.

B) let *geometry* be the *ST_Point* value which specifies the node location.

C) insert into *ST_TOPO_NET*.*ST_NODE* values (*anetwork*, *nodeid*, *geometry*).

3) for each link:

A) let *linkid* be the generated unique link id INTEGER value.

B) let *startnode* be the INTEGER value equal to the node id of the node at the start of the link.

C) let *endnode* be the INTEGER value equal to the node id of the node at the end of the link.

D) let *geometry* be the *ST_Curve* value which represents the geometry of the link, in the same direction as the link.

E) insert into *ST_TOPO_NET*.*ST_LINK* values (*anetwork*, *linkid*, *startnode*, *endnode*, *geometry*).

3) All non-null geometry values in the *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* base tables in rows which have the same *NETWORK* column value shall have the same spatial reference system identifier.

12.3.17 ST_ValidLogicalNet Function

Purpose

Return a table containing possible network inconsistencies for a logical topology-network.

Definition

```
CREATE FUNCTION ST_ValidLogicalNet
   (anetwork CHARACTER VARYING(ST_MaxNetworkName))
   RETURNS TABLE
      (error CHARACTER VARYING(30),
       primitive1 INTEGER,
       primitive2 INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_ValidLogicalNet(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*.
2) For the function *ST_ValidLogicalNet(CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:

      i) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_LINK* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – empty network*.
      ii) Otherwise,
         1) let T be a table value consisting of the following three columns:
            A) a column *error* of type CHARACTER VARYING, which identifies the type of inconsistency found in *anetwork*.
            B) a column *primitive1* of type INTEGER, which contains the node or link id of the first offending network primitive.
            C) a column *primitive2* of type INTEGER, which contains the node or link id of the second offending network primitive.

2) for each node with a non-null geometry value:

   A) let *E* be the CHARACTER VARYING value equal to 'node has geometry'.

   B) let *N1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_NODE*.*NODE_ID* for a node in *anetwork*.

   C) let *G1* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N1*.

   D) let *N2* be equal to the null value.

   E) for all values of *N1* such that *G1* is not equal to the null value, insert into *T* values (*E*, *N1*, *N2*).

3) for each link with a non-null geometry value:

   A) let *E* be the CHARACTER VARYING value equal to 'link has geometry'.

   B) let *L1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*.*LINK_ID* for a link in *anetwork*.

   C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *L1*.

   D) let *L2* be equal to the null value.

   E) for all values of *L1* such that *G1* is not equal to the null value, insert into *T* values (*E*, *L1*, *L2*).

12.3.18 ST_ValidSpatialNet Function

Purpose

Return a table containing network inconsistencies for a spatial topology-network.

Definition

```
CREATE FUNCTION ST_ValidSpatialNet
   (atopology CHARACTER VARYING(ST_MaxNetworkName))
   RETURNS TABLE
      (error CHARACTER VARYING(30),
       primitive1 INTEGER,
       primitive2 INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

Description

1) The function *ST_ValidSpatialNet(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork*.
2) For the function *ST_ValidSpatialNet(CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anetwork* is not a valid schema name, then an exception condition is raised: *SQL/MM Spatial exception – invalid network name*.
   c) If the user has not been granted SELECT privilege on *anetwork*.*ST_NODE*, then an exception condition is raised: *SQL/MM Spatial exception – network privilege denied*.
   d) Otherwise, for rows in *ST_TOPO_NET*.*ST_NODE* and *ST_TOPO_NET*.*ST_LINK* where *NETWORK* is equal to *anetwork*:

      Case:

      i) If SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_NODE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *ST_TOPO_NET*.*ST_LINK* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – empty network*.
      ii) Otherwise,
         1) let *T* be a table value consisting of the following three columns:
            A) a column *error* of type CHARACTER VARYING, which identifies the type of inconsistency found in *anetwork*.
            B) a column *primitive1* of type INTEGER, which contains the node or link id of the first offending network primitive.
            C) a column *primitive2* of type INTEGER, which contains the node or link id of the second offending network primitive.

2) for each node with a null geometry value:
   A) let *E* be the CHARACTER VARYING value equal to 'missing node geometry'.
   B) let *N1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_NODE*.*NODE_ID* for a node in *anetwork*.
   C) let *G1* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N1*.
   D) let *N2* be equal to the null value.
   E) for all values of *N1* such that *G1* is equal to the null value, insert into *T* values (*E*, *N1*, *N2*).
3) for each link with a null geometry value:
   A) let *E* be the CHARACTER VARYING value equal to 'missing link geometry'.
   B) let *L1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*.*LINK_ID* for a link in *anetwork*.
   C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *L1*.
   D) let *L2* be equal to the null value.
   E) for all values of *L1* such that *G1* is equal to the null value, insert into *T* values (*E*, *L1*, *L2*),.
4) for each link having a geometry with a start point not equal to the geometry of its start node:
   A) let *E* be the CHARACTER VARYING value equal to 'geometry start mismatch'.
   B) let *L1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*_ID for a link in *anetwork*.
   C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *L1*.
   D) let *N2* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*.*START_NODE* where *LINK_ID* is equal to *L1*.
   E) let *G2* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.
   F) for all values of *L1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_StartPoint())* is not equal to 1 (one), insert into *T* values (*E*, *L1*, *N2*).
5) for each link having a geometry with an end point not equal to the geometry of its end node:
   A) let *E* be the CHARACTER VARYING value equal to 'geometry end mismatch'.
   B) let *L1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*_ID for a link in *anetwork*.
   C) let *G1* be the *ST_Curve* value equal to *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *L1*.
   D) let *N2* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*.*END_NODE* where *LINK_ID* is equal to *L1*.
   E) let *G2* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.
   F) for all values of *L1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_EndPoint())* is not equal to 1 (one), insert into *T* values (*E*, *L1*, *N2*).
6) if all geometries do not have the same spatial reference system identifier:

A) let *E* be the CHARACTER VARYING value equal to 'mixed SRIDs'.

B) let *N1* be the INTEGER value equal to *ST_TOPO_NET*.*ST_NODE*.*NODE_ID* for a node in *anetwork*.

C) let *G1* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N1*.

D) let *S1* be the INTEGER value returned by *G1.ST_SRID()*.

E) let *N2* be the INTEGER value equal to *ST_TOPO_NET*.*ST_NODE*.*NODE_ID* for another node in *anetwork*.

F) let *G2* be the *ST_Point* value equal to *ST_TOPO_NET*.*ST_NODE*.*GEOMETRY* where *NODE_ID* is equal to *N2*.

G) let *S2* be the INTEGER value returned by *G2.ST_SRID()*.

H) let *L3* be the INTEGER value equal to *ST_TOPO_NET*.*ST_LINK*.*LINK_ID* for a link in *anetwork*.

I) let *G3* be the *ST_Curve* value equal to *ST_TOPO_NET*.*ST_LINK*.*GEOMETRY* where *LINK_ID* is equal to *L3*.

J) let *S3* be the INTEGER value returned by *G3.ST_SRID()*.

K) if any value of *N2* or *L3* has a corresponding value of *S2* or *S3*, respectively, such that *S2* is not equal to *S1* or *S3* is not equal to *S1*, insert into *T* values (*E*, NULL, NULL).

13 General Routines

13.1 Shortest Path Routines

13.1.1 ST_ShortestUndPath Function

Purpose

Return a table containing IDs of undirected shortest paths between two specified points that shall be either a start or end point of simple ST_Geometry value in a referenced table.

Definition

```
CREATE FUNCTION ST_ShortestUndPath
   (paths_table_name DT1,
    path_id_column DT2,
    geometry_column DT2,
    edge_weight_column DT2,
    start_point ST_Point,
    end_point ST_Point)
   RETURNS TABLE
      (shortest_path INTEGER ARRAY[ST_MaxArrayElements],
       total_weight DOUBLE PRECISION)
   DETERMINISTIC
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE FUNCTION ST_ShortestUndPath
   (paths_table_name DT1,
    path_id_column DT2,
    geometry_column DT2,
    start_point ST_Point,
    end_point ST_Point)
   RETURNS TABLE
      (shortest_path INTEGER ARRAY[ST_MaxArrayElements],
       total_weight DOUBLE PRECISION)
   DETERMINISTIC
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxArrayElements* is the implementation-defined maximum cardinality of an array for the number of geometric paths.
2) *DT1* is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.
3) *DT2* is data type of variable-length character string with character set SQL_IDENTIFIER and maximum length not less than 128 characters.

Description

1) The ST_ShortestUndPath functions are for public use.
2) The function *ST_ShortestUndPath* takes the following input parameters:

a) a value *paths_table_name* of type *DT1*, which has the name of a referenced table consisting of at least three columns: an INTEGER type column which has a unique number to identify a path as path identifier, an *ST_Geometry* type column which has a spatial representation of a path with 1-dimensional geometry as path geometry, and a DOUBLE PRECISION type column which has weight value on the path as edge weight. Let *S* be *paths_table_name* value. Let *V* be the character string that is the value of TRIM(BOTH ' ' FROM *S*). If *V* value does not conform to the Format and Syntax Rules of <table name> specified in ISO/IEC 9075-2, or the table specified by paths_table_name value does not exist in the system, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

b) a value *path_id_column* of type *DT2*, which has the name of the column for path identifier in the table specified by *paths_table_name*. If the column specified by *path_id_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

c) a value *geometry_column* of type *DT2*, which has the name of the column for path geometry in the table specified by *paths_table_name*. If the column specified by *geometry_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

d) a value *edge_weight_column* of type *DT2*, which has the name of the column for edge weight in the table specified by *paths_table_name*. If the column specified by *edge_weight_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

e) an *ST_Point* value *start_point*, which is a start point for getting the shortest geometric paths. If *start_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

f) an *ST_Point* value *end_point*, which is an end point for acquiring the shortest geometric paths. If *end_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) The function *ST_ShortestUndPath* takes the following input parameters:

a) a value *paths_table_name* of type *DT1*, which has the name of a referenced table consisting of at least two columns:an INTEGER type column which has a unique number to identify a path as path identifier, and an *ST_Geometry* type column which has a spatial representation of a path with 1-dimensional geometry as path geometry. Let *S* be *paths_table_name* value. Let *V* be the character string that is the value of TRIM(BOTH ' ' FROM S). If *V* value does not conform to the Format and Syntax Rules of <table name> specified in ISO/IEC 9075-2, or the table specified by *paths_table_name* value does not exist in the system, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

b) a value *path_id_column* of type *DT2*, which has the name of the column for path identifier in the table specified by *paths_table_name*. If the column specified by *path_id_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

c) a value *geometry_column* of type *DT2*, which has the name of the column for path geometry in the table specified by *paths_table_name*. If the column specified by *geometry_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

d) an *ST_Point* value *start_point*, which is a start point for getting the shortest geometric paths. If *start_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

e) an *ST_Point* value *end_point*, which is an end point for acquiring the shortest geometric paths. If *end_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

4) The function *ST_ShortestUndPath* with input parameters (*paths_table_name*, *path_id_column*, *geometry_column*, *edge_weight_column*, *start_point*, *end_point*) returns the following value:

a) a table value consists of the following two columns:

i) a column *shortest_path* of type ARRAY of INTEGER, which has a representation of shortest geometric paths using edge weight as geometric length from the start point to the end point by a sequence of path identifier defined by 2) a).

ii) a column *total_weight* of type DOUBLE PRECISION, which has the total geometric length of *shortest_path*.

5) The function *ST_ShortestUndPath* with input parameters (*paths_table_name*, *path_id_column*, *geometry_column*, *start_point*, *end_point*) returns the following value:

a) a table value consists of the following two columns:

i) a column *shortest_path* of type ARRAY of INTEGER, which has a representation of shortest geometric paths using length value of path geometry from the start point to the end point by a sequence of path identifier defined by 3) a).

ii) a column *total_weight* of type DOUBLE PRECISION, which has the total length of path geometries of *shortest_path*.

6) Case:

a) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_ShortestUndPath* is in the linear unit of measure identified by <linear unit>.

b) Otherwise, the value returned by *ST_ShortestUndPath* is in an implementation-defined unit of measure.

7) If the values in the column specified by *path_id_column* in the table specified by *paths_table_name* are not unique values, then an exception condition is raised: *SQL/MM Spatial exception – duplicate value*.

8) If the column specified by *geometry_column* contains a value of subtypes other than *ST_Curve*, then the row of the value shall be ignored.

9) If the table specified by *paths_table_name* has no rows, then the function *ST_ShortestUndPath* returns no rows.

10) If no contiguous paths is found from the start point to the end point in the table specified by *paths_table_name*, then the function *ST_ShortestUndPath* returns no rows.

11) If there are one or more shortest paths that have the same total length, then the function *ST_ShortestUndPath* returns the rows of those plural shortest paths.

13.1.2 ST_ShortestDirPath Function

Purpose

Return a table containing IDs of directed shortest paths between two specified points of simple ST_Geometry value in a referenced table.

Definition

```
CREATE FUNCTION ST_ShortestDirPath
   (paths_table_name DT1,
    path_id_column DT2,
    geometry_column DT2,
    path_start_column DT2,
    edge_weight_column DT2,
    start_point ST_Point,
    end_point ST_Point)
   RETURNS TABLE
      (shortest_path INTEGER ARRAY[ST_MaxArrayElements],
       total_weight DOUBLE PRECISION)
   DETERMINISTIC
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE FUNCTION ST_ShortestDirPath
   (paths_table_name DT1,
    path_id_column DT2,
    geometry_column DT2,
    path_start_column DT2,
    start_point ST_Point,
    end_point ST_Point)
   RETURNS TABLE
      (shortest_path INTEGER ARRAY[ST_MaxArrayElements],
       total_weight DOUBLE PRECISION )
   DETERMINISTIC
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxArrayElements* is the implementation-defined maximum cardinality of an array for the number of geometric paths.
2) *DT1* is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.
3) *DT2* is data type of variable-length character string with character set SQL_IDENTIFIER and maximum length not less than 128 characters.

Description

1) The ST_ShortestDirPath functions are for public use.
2) The function *ST_ShortestDirPath* takes the following input parameters:

a) a value *paths_table_name* of type *DT1*, which has the name of a referenced table consisting of at least four columns: an INTEGER type column which has a unique number to identify a path as path identifier, an *ST_Geometry* type column which has a spatial representation of a path with 1-dimensional geometry as path geometry, an *ST_Point* type column which is a start point of the path specified by path identifier and a DOUBLE PRECISION type column which has weight value on the path as edge weight. Let *S* be paths_table_name value. Let *V* be the character string that is the value of TRIM(BOTH ' ' FROM S). If *V* value does not conform to the Format and Syntax Rules of <table name> specified in ISO/IEC 9075-2, or the table specified by *paths_table_name* value does not exist in the system, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

b) a value *path_id_column* of type *DT2*, which has the name of the column for path identifier in the table specified by *paths_table_name*. If the column specified by *path_id_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

c) a value *geometry_column* of type *DT2*, which has the name of the column for path geometry in the table specified by *paths_table_name*. If the column specified by *geometry_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

d) a value *path_start_column* of type *DT2*, which has the name of the column for start point of path in the table specified by *paths_table_name*. If the column specified by *path_start_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

e) a value *edge_weight_column* of type *DT2*, which has the name of the column for edge weight in the table specified by *paths_table_name*. If the column specified by *edge_weight_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

f) an *ST_Point* value *start_point* , which is a start point for getting the shortest geometric paths. If the *ST_Point* value *start_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

g) an *ST_Point* value *end_point* , which is an end point for acquiring the shortest geometric paths. If the *ST_Point* value *end_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) The function *ST_ShortestDirPath* takes the following input parameters:

a) a value *paths_table_name* of type *DT1*, which has the name of a referenced table consisting of at least four columns: an INTEGER type column which has a unique number to identify a path as path identifier, an *ST_Geometry* type column which has a spatial representation of a path with 1-dimensional geometry as path geometry, and an *ST_Point* type column which is a start point of the path specified by path identifier. Let *S* be paths_table_name value. Let *V* be the character string that is the value of TRIM(BOTH ' ' FROM S). If *V* value does not conform to the Format and Syntax Rules of <table name> specified in ISO/IEC 9075-2, or the table specified by *paths_table_name* value does not exist in the system, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

b) a value *path_id_column* of type *DT2*, which has the name of the column for path identifier in the table specified by *paths_table_name*. If the column specified by *path_id_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

c) a value *geometry_column* of type *DT2*, which has the name of the column for path geometry in the table specified by *paths_table_name*. If the column specified by *geometry_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

d) a value *path_start_column* of type *DT2*, which has the name of the column for start point of path in the table specified by *paths_table_name*. If the column specified by *path_start_column* does not exist in the table specified by *paths_table_name*, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

e) an *ST_Point* value *start_point*, which is a start point for getting the shortest geometric paths. If the *ST_Point* value *start_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

f) an *ST_Point* value *end_point*, which is an end point for acquiring the shortest geometric paths. If the *ST_Point* value *end_point* is not one of the points in path geometry, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

4) The function *ST_ShortestDirPath* with input parameters(*paths_table_name*, *path_id_column*, *geometry_column*, *path_start_column*, *edge_weight_column*, *start_point*, *end_point*) returns the following value:

a) a table value consists of the following two columns:

i) a column *shortest_path* of type ARRAY of INTEGER, which has a representation of shortest geometric paths using edge weight as geometric length from the start point to the end point by a sequence of path identifier defined by 2) a).

ii) a column *total_weight* of type DOUBLE PRECISION, which has the total geometric length of *shortest_path*.

5) The function *ST_ShortestDirPath* with input parameters(*paths_table_name*, *path_id_column*, *geometry_column*, *path_start_column*, *start_point*, *end_point*) returns the following value:

a) a table value consists of the following two columns:

i) a column *shortest_path* of type ARRAY of INTEGER, which has a representation of shortest geometric paths using length value of path geometry from the start point to the end point by a sequence of path identifier defined by 3) a).

ii) a column *total_weight* of type DOUBLE PRECISION, which has the total length value of path geometries of *shortest_path*.

6) Case:

a) If the spatial reference system of SELF defines a <linear unit>, then the value returned by *ST_ShortestDirPath* is in the linear unit of measure identified by <linear unit>.

b) Otherwise, the value returned by *ST_ShortestDirPath* is in an implementation-defined unit of measure.

7) if the values in the column specified by *path_id_column* in the table specified by *paths_table_name* are not unique values, then an exception condition is raised: *SQL/MM Spatial exception – duplicate value*.

8) if the column specified by *geometry_column* contains a value of subtypes other than *ST_Curve*, then the row of the value shall be ignored.

9) if the table specified by *paths_table_name* has no row, then the function *ST_ShortestDirPath* returns no row.

10) if no contiguous path is found from the start point to the end point in the table specified by *paths_table_name*, then the function *ST_ShortestDirPath* returns no row.

11) if there are one or more shortest paths that have the same total length, then the function *ST_ShortestDirPath* returns the rows of those plural shortest paths.

14 Spatial Reference System Type

14.1 ST_SpatialRefSys Type and Routines

14.1.1 ST_SpatialRefSys Type

Purpose

The ST_SpatialRefSys type encapsulates all aspects of spatial reference systems.

Definition

```
CREATE TYPE ST_SpatialRefSys
   AS (
      --
      -- See Description
      --
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_SpatialRefSys
      (awkt CHARACTER LARGE OBJECT(ST_MaxSRSAsText))
      RETURNS ST_SpatialRefSys
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_SpatialRefSys
      (ansrid INTEGER)
      RETURNS ST_SpatialRefSys
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_AsWKTSRS()
      RETURNS CHARACTER LARGE OBJECT(ST_MaxSRSAsText)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_WKTSRSToSQL
      (awkt CHARACTER LARGE OBJECT(ST_MaxSRSAsText))
      RETURNS ST_SpatialRefSys
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_SRID()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Equals
   (ansrs ST_SpatialRefSys)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxSRSAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_SpatialRefSys* value.

Description

1) The *ST_SpatialRefSys* type provides for public use:
   a) a method *ST_SpatialRefSys(CHARACTER LARGE OBJECT)*,
   b) a method *ST_SpatialRefSys(INTEGER)*,
   c) a method *ST_AsWKTSRS()*,
   d) a method *ST_WKTSRSToSQL(CHARACTER LARGE OBJECT)*,
   e) a method *ST_SRID()*,
   f) a method *ST_Equals(ST_SpatialRefSys)*,
   g) an ordering function *ST_OrderingEquals(ST_SpatialRefSys, ST_SpatialRefSys)*,
   h) an SQL Transform group *ST_WellKnownText*.
2) The attribute definitions in the *ST_SpatialRefSys* type are implementation-dependent.

   NOTE Implementations should refer to ISO 19111 as a model to follow for the implementation-dependent attribute definitions in the ST_SpatialRefSys type.

14.1.2 ST_SpatialRefSys Methods

Purpose

Return a specified ST_SpatialRefSys value.

Definition

```
CREATE CONSTRUCTOR METHOD ST_SpatialRefSys
   (awkt CHARACTER LARGE OBJECT(ST_MaxSRSAsText))
   RETURNS ST_SpatialRefSys
   FOR ST_SpatialRefSys
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_SpatialRefSys
   (ansrid INTEGER)
   RETURNS ST_SpatialRefSys
   FOR ST_SpatialRefSys
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxSRSAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_SpatialRefSys* value.

Description

1) The method *ST_SpatialRefSys(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The parameter *awkt* is the well-known text representation of an *ST_SpatialRefSys* value. If *awkt* is not producible in the BNF for <spatial reference system>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
3) The null-call type-preserving SQL-invoked constructor method *ST_SpatialRefSys(CHARACTER LARGE OBJECT)* returns an *ST_SpatialRefSys* value representing the spatial reference system defined by *awkt*.
4) The method *ST_SpatialRefSys(INTEGER)* takes the following input parameters:
   a) an INTEGER value *ansrid*.
5) The null-call type-preserving SQL-invoked constructor method *ST_SpatialRefSys(INTEGER)* returns an *ST_SpatialRefSys* value representing the spatial reference system defined by the spatial reference system identifier, *ansrid*.

14.1.3 ST_AsWKTSRS Method

Purpose

Return the well-known text representation of a spatial reference system.

Definition

```
CREATE METHOD ST_AsWKTSRS()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxSRSAsText)
   FOR ST_SpatialRefSys
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxSRSAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_SpatialRefSys* value.

Description

1) The method *ST_AsWKTSRS()* has no input parameters.
2) The null-call method *ST_AsWKTSRS()* returns a CHARACTER LARGE OBJECT value containing the well-known text representation of SELF. Values shall be produced in the BNF for <spatial reference system>.

14.1.4 ST_WKTSRSToSQL Method

Purpose

Return the ST_SpatialRefSys value repsentedby the specified well-known text representation for a spatial reference system.

Definition

```
CREATE METHOD ST_WKTSRSToSQL
   (awkt CHARACTER LARGE OBJECT(ST_MaxSRSAsText))
   RETURNS ST_SpatialRefSys
   FOR ST_SpatialRefSys
   RETURN SELF.ST_SpatialRefSys(awkt)
```

Definitional Rules

1) *ST_MaxSRSAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_SpatialRefSys*.

Description

1) The method *ST_WKTSRSToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT *awkt*.
2) The parameter *awkt* is the well-known text representation of an *ST_SpatialRefSys* value. If *awkt* is not producible in the BNF for <spatial reference system>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
3) The null-call method *ST_WKTSRSToSQL(CHARACTER LARGE OBJECT)* returns an *ST_SpatialRefSys* value represented by *awkt.*

14.1.5 ST_SRID Method

Purpose

Return a spatial reference system identifier of an ST_SpatialRefSys value.

Definition

```
CREATE METHOD ST_SRID()
   RETURNS INTEGER
   FOR ST_SpatialRefSys
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_SRID()* has no input parameters.
2) The null-call method *ST_SRID()* returns an INTEGER value representing a unique identifier. This unique identifier is called the *spatial reference system identifier*. A spatial reference system identifier that is equal to 0 (zero) is implementation-defined. A spatial reference system identifier that is not equal to 0 (zero) is implementation-dependent.

14.1.6 ST_Equals Method

Purpose

Test if two ST_SpatialRefSys values are equal.

Definition

```
CREATE METHOD ST_Equals
   (ansrs ST_SpatialRefSys)
   RETURNS INTEGER
   FOR ST_SpatialRefSys
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Equals(ST_SpatialRefSys)* takes the following input parameters:
   a) an *ST_SpatialRefSys* value *ansrs*.
2) For the null-call method *ST_Equals(ST_SpatialRefSys)*:

   Case:

   a) If SELF is equal to *ansrs*, then return 1 (one).
   b) Otherwise, return 0 (zero).
3) The method *ST_Equals(ST_SpatialRefSys)* is implementation-defined.

14.1.7 ST_OrderingEquals Function

Purpose

Define the equals ordering for the ST_SpatialRefSys type.

Definition

```
CREATE FUNCTION ST_OrderingEquals
   (ansrs ST_SpatialRefSys,
    anothersrs ST_SpatialRefSys)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN
      CASE
         WHEN ansrs.ST_Equals(anothersrs) = 1 THEN
            0
         ELSE
            1
      END

CREATE ORDERING FOR ST_SpatialRefSys
   EQUALS ONLY BY RELATIVE
      WITH FUNCTION ST_OrderingEquals(ST_SpatialRefSys, ST_SpatialRefSys)
```

Description

1) The function *ST_OrderingEquals(ST_SpatialRefSys, ST_SpatialRefSys)* takes the following input parameters:
   a) an *ST_SpatialRefSys* value *ansrs*,
   b) an *ST_SpatialRefSys* value *anothersrs*.
2) For the null-call function *ST_OrderingEquals(ST_SpatialRefSys, ST_SpatialRefSys)*:

   Case:
   a) If the value expression *ansrs*.*ST_Equals(anothersrs)* is 1 (one), then return 0 (zero).
   b) Otherwise, return 1 (one).
3) Use the function *ST_OrderingEquals(ST_SpatialRefSys, ST_SpatialRefSys)* to define ordering for the *ST_SpatialRefSys* type.

14.1.8 ST_WellKnownText SQL Transform Group

Purpose

Define SQL transform functions for the ST_SpatialRefSys type.

Definition

```
CREATE TRANSFORM FOR ST_SpatialRefSys
   ST_WellKnownText
      (TO SQL WITH METHOD ST_WKTSRSToSQL
         (CHARACTER LARGE OBJECT(ST_MaxSRSAsText)),
       FROM SQL WITH METHOD ST_AsWKTSRS())
```

Definitional Rules

1) *ST_MaxSRSAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_SpatialRefSys*.

Description

1) Use the method *ST_WKTSRSToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsWKTSRS()* to define the transform group *ST_WellKnownText*.

14.1.9 <spatial reference system>

Purpose

This subclause contains the definition of <spatial reference system>.

Description

1) The well-known text representation of an *ST_SpatialRefSys* value is defined by the following BNF for <spatial reference system>.

```
<spatial reference system> ::=
     <projected cs>
   | <geographic cs>
   | <geocentric cs>

<projected cs> ::=
     PROJCS <left delimiter>
        <double quote> <name> <double quote> <comma>
        <geographic cs> <comma>
        <projection> <comma>
        { <parameter> <comma> }...
        <linear unit>
        <right delimiter>

<geographic cs> ::=
     GEOGCS <left delimiter>
        <double quote> <name> <double quote> <comma>
        <datum> <comma>
        <prime meridian> <comma>
        <angular unit>
        [ <linear unit> ]
        <right delimiter>

<geocentric cs> ::=
     GEOCCS <left delimiter>
        <double quote> <name> <double quote> <comma>
        <datum> <comma>
        <prime meridian> <comma>
        <linear unit>
        <right delimiter>

<projection> ::=
     PROJECTION <left delimiter>
        <double quote> <projection name> <double quote>
        <right delimiter>

<parameter> ::=
     PARAMETER <left delimiter>
        <double quote> <parameter name> <double quote> <comma>
        <value>
        <right delimiter>

<value> ::= <number>

<datum> ::=
     DATUM <left delimiter>
        <double quote> <datum name> <double quote> <comma>
        <spheroid>
        <right delimiter>
```

```
<spheroid> ::=
     SPHEROID <left delimiter>
        <double quote> <spheroid name> <double quote> <comma>
        <semi-major axis> <comma>
        <inverse flattening>
        <right delimiter>

<semi-major axis> ::= <number>

<inverse flattening> ::= <number>

<prime meridian> ::=
     PRIMEM <left delimiter>
        <double quote> <prime meridian name> <double quote> <comma>
        <longitude>
        <right delimiter>

<longitude> ::= <number>

<angular unit> ::= <unit>

<linear unit> ::= <unit>

<unit> ::=
     UNIT <left delimiter>
        <double quote> <unit name> <double quote> <comma>
        <conversion factor>
        <right delimiter>

<conversion factor> ::= <number>

<datum name> ::= <letters>

<parameter name> ::= <letters>

<prime meridian name> ::= <letters>

<projection name> ::= <letters>

<spheroid name> ::= <letters>

<unit name> ::= <letters>

<name> ::= <letters>

<letters> ::= <letter>...

<letter> ::=
     <simple Latin letter>
   | <digit>
   | <special>

<special> ::=
     <left paren>
   | <right paren>
   | <minus sign>
   | <underscore>
   | <period>
   | <quote>
   | <space>

<number> ::=
     <exact numeric literal>
   | <approximate numeric literal>

<left delimiter> ::=
     <left paren>
   | <left bracket>
```

```
<right delimiter> ::=
     <right paren>
   | <right bracket>

<exact numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<approximate numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<simple Latin letter> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<digit> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<double quote> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<comma> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<left bracket> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right bracket> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<left paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<minus sign> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<underscore> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<period> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<quote> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<space> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075
```

a) Case:

   i) If <left paren> is used as a <left delimiter>, then <right paren> shall be used as the corresponding <right delimiter>.

   ii) If <left bracket> is used as a <left delimiter>, then <right bracket> shall be used as the corresponding <right delimiter>.

b) A <spatial reference system> is a geographic (latitude-longitude), a projected (X, Y), or a geocentric (X, Y, Z) coordinate system.

The coordinate reference system support for *ST_Geometry* values with m coordinate values is implementation-defined.

The coordinate system is composed of several items. Each item has a keyword in upper case followed by the defining, comma-delimited, parameters of the item in brackets. Some items are composed of other items in a nested structure.

c) The list of keywords are DATUM, GEOCCS, GEOGCS, PROJCS, PARAMETER, PRIMEM, PROJECTION, SPHEROID, and UNIT.

d) <projected cs> is a projected coordinate system. <projection name> is an implementation-defined name of a parameter.

e) <geographic cs> is a geographic coordinate system.

f) <geocentric cs> is a geocentric coordinate system.

g) <name> is the name of the coordinate system.

h) <projection> is the projection system of a <projected cs>.

i) <parameter> is a parameter of the <projection> to define the <projected cs>. <parameter name> is an implementation-defined name of a parameter.

j) <datum> is the horizontal datum used by the <geographic cs> or the <geocentric cs>. <datum name> is an implementation-defined name of a datum.

k) <spheroid> is the spheroid of a datum defined by a semi-major axis, <semi-major axis> , and an inverse flattening, <inverse flattening>. <spheroid>s are implementation-defined.

l) <prime meridian> is the longitude used by the <geographic cs> or the <geocentric cs>. The <semi-major axis> is greater than 0 (zero) and it is measured in meters. <prime meridian>s are implementation-defined.

m) <angular unit> specifies an angular unit and <linear unit> defines a linear unit. <conversion factor> specifies the number of meters for a linear unit or the number of radians for an angular unit per unit. <conversion factor> is greater than zero. <angular unit>s and <linear unit>s are implementation-defined.

15 Linear Referencing Types

15.1 ST_LRM Type and Routines

15.1.1 ST_LRM Type

Purpose

The ST_LRM type specifies the Linear Referencing Method which describes the manner in which measurements are made along (and optionally offset from) a linear element. The ST_LRM type is instantiable.

Definition

```
CREATE TYPE ST_LRM
   AS (
      ST_PrivateLRMID INTEGER DEFAULT NULL,
      ST_PrivateLRMName CHARACTER VARYING(ST_MaxLRMNameLength)
         DEFAULT NULL,
      ST_PrivateLRMType CHARACTER VARYING(128) DEFAULT NULL,
      ST_PrivateUnits CHARACTER VARYING(ST_MaxUnitNameLength)
         DEFAULT NULL,
      ST_PrivateConstraints CHARACTER VARYING(ST_MaxConstraintLength)
         ARRAY[ST_MaxConstraintArrayElements] DEFAULT ARRAY[],
      ST_PrivateOffsetUnits CHARACTER VARYING(ST_MaxUnitNameLength)
         DEFAULT NULL,
      ST_PrivatePositiveLateralOffsetDirection CHARACTER VARYING(10)
         DEFAULT NULL,
      ST_PrivatePositiveVerticalOffsetDirection CHARACTER VARYING(10)
         DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LRM
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
      RETURNS ST_LRM
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LRM
      (anlrmid INTEGER,
       anlrmname CHARACTER VARYING(ST_MaxLRMNameLength),
       anlrmtype CHARACTER VARYING(128),
       aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength),
       aconstraintarray CHARACTER VARYING(ST_MaxConstraintLength)
         ARRAY[ST_MaxConstraintArrayElements])
      RETURNS ST_LRM
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_LRM
   (anlrmid INTEGER,
    anlrmname CHARACTER VARYING(ST_MaxLRMNameLength),
    anlrmtype CHARACTER VARYING(128),
    aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength),
    aconstraintarray CHARACTER VARYING(ST_MaxConstraintLength)
       ARRAY[ST_MaxConstraintArrayElements],
    anoffsetunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength),
    apositivelateraloffsetdirection CHARACTER VARYING(10),
    apositiveverticaloffsetdirection CHARACTER VARYING(10))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LRMID()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LRMID
   (anlrmid INTEGER)
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LRMName()
   RETURNS CHARACTER VARYING(ST_MaxLRMNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LRMName
   (anlrmname CHARACTER VARYING(ST_MaxLRMNameLength))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LRMType()
   RETURNS CHARACTER VARYING(128)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_LRMType
   (anlrmtype CHARACTER VARYING(128))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_UnitOfMeasure()
   RETURNS CHARACTER VARYING(ST_MaxUnitNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_UnitOfMeasure
   (aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Constraints()
   RETURNS CHARACTER VARYING(ST_MaxConstraintLength)
      ARRAY[ST_MaxConstraintArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Constraints
   (aconstraintarray CHARACTER VARYING(ST_MaxConstraintLength)
      ARRAY[ST_MaxConstraintArrayElements])
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_OffsetMeasUnit()
   RETURNS CHARACTER VARYING(ST_MaxUnitNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_OffsetMeasUnit
   (anoffsetunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_PosLatOffsetDir()
   RETURNS CHARACTER VARYING(10)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PosLatOffsetDir
   (apositivelateraloffsetdirection CHARACTER VARYING(10))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_PosVerOffsetDir()
   RETURNS CHARACTER VARYING(10)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_PosVerOffsetDir
   (apositiveverticaloffsetdirection CHARACTER VARYING(10))
   RETURNS ST_LRM
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxConstraintArrayElements* is the implementation-defined maximum cardinality of an array of CHARACTER VARYING constraint values.
2) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
3) *ST_MaxLRMNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a linear referencing method.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
5) *ST_MaxConstraintLength* is the implementation-defined maximum length of the CHARACTER VARYING used for an LRM constraint.
6) The attribute *ST_PrivateLRMID is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLRMID*.
7) The attribute *ST_PrivateLRMName is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLRMName*.
8) The attribute *ST_PrivateLRMType is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLRMType*.
9) The attribute *ST_PrivateUnits is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateUnits*.
10) The attribute *ST_PrivateConstraints is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateConstraints*.
11) The attribute *ST_PrivateOffsetUnits is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateOffsetUnits*.

12) The attribute *ST_PrivatePositiveLateralOffsetDirection* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePositiveLateralOffsetDirection*.

13) The attribute *ST_PrivatePositiveVerticalOffsetDirection* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePositiveVerticalOffsetDirection*.

Description

1) The *ST_LRM* type provides for public use:
   a) a method *ST_LRM(CHARACTER LARGE OBJECT)*,
   b) a method *ST_LRM(INTEGER, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING ARRAY)*,
   c) a method *ST_LRM(INTEGER, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING ARRAY, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING)*,
   d) a method *ST_LRMID()*,
   e) a method *ST_LRMID(INTEGER)*,
   f) a method *ST_LRMName()*,
   g) a method *ST_LRMName(CHARACTER VARYING)*,
   h) a method *ST_LRMType()*,
   i) a method *ST_LRMType(CHARACTER VARYING)*,
   j) a method *ST_UnitOfMeasure()*,
   k) a method *ST_UnitOfMeasure(CHARACTER VARYING)*,
   l) a method *ST_Constraints()*,
   m) a method *ST_Constraints(CHARACTER VARYING ARRAY)*,
   n) a method *ST_OffsetMeasUnit()*,
   o) a method *ST_OffsetMeasUnit(CHARACTER VARYING)*,
   p) a method *ST_PosLatOffsetDir()*,
   q) a method *ST_PosLatOffsetDir(CHARACTER VARYING)*,
   r) a method *ST_PosVerOffsetDir()*,
   s) a method *ST_PosVerOffsetDir(CHARACTER VARYING)*,
   t) a function *ST_LRMFromText(CHARACTER LARGE OBJECT)*,
   u) a function *ST_LRMFromGML(CHARACTER LARGE OBJECT).*

2) The *ST_PrivateLRMID* attribute contains the INTEGER lrmid identifier of the Linear Referencing Method.

3) The *ST_PrivateLRMName* attribute contains the CHARACTER VARYING Linear Referencing Method name value.

4) The *ST_PrivateLRMType* attribute contains the CHARACTER VARYING Linear Referencing Method type value.

5) The *ST_PrivateUnits* attribute contains the CHARACTER VARYING Linear Referencing Method unit of measure value.

6) The *ST_PrivateConstraints* attribute contains the optional CHARACTER VARYING ARRAY collection of Linear Referencing Method constraint values.

7) The *ST_PrivateOffsetUnits* attribute contains the optional CHARACTER VARYING Linear Referencing Method offset unit of measure value.

8) The *ST_PrivatePositiveLateralOffsetDirection* attribute contains the optional CHARACTER VARYING Linear Referencing Method positive lateral offset direction value.
9) The *ST_PrivatePositiveVerticalOffsetDirection* attribute contains the optional CHARACTER VARYING Linear Referencing Method positive vertical offset direction value.
10) If the *ST_LRM* allows offsets, then *ST_PrivateOffsetUnits, ST_PrivatePositiveLateralOffsetDirection* and *ST_PrivatePositiveVerticalOffsetDirection* shall not be NULL and the default values for *ST_PrivatePositiveLateralOffsetDirection* and *ST_PrivatePositiveVerticalOffsetDirection* shall be 'right' and 'up', respectively. Otherwise, all three attributes shall be NULL. Specifying a NULL value for *ST_PrivateOffsetUnits* will cause the other two attributes to be set to NULL.
11) This lateral and vertical offset directions are as viewed from above the linear element facing in the direction of increasing measure. If "from" and "towards referents" have been specified, then the offset direction is as viewed from above the "from referent" facing in the direction of the "towards referent".
12) The lrmid identifier specified by the *ST_PrivateLRMID* attribute shall be unique across all Linear Referencing Methods.
13) The allowable LRM type values specified by the *ST_PrivateLRMType* attribute shall include 'absolute', 'relative', 'interpolative', and 'local interpolative'.
14) The allowable positive lateral offset direction values specified by the *ST_PrivatePositiveLateralOffsetDirection* attribute shall include 'right' and 'left'.
15) The allowable positive vertical offset direction values specified by the *ST_PrivatePositiveVerticalOffsetDirection* attribute shall include 'up' and 'down'.
16) The allowable values specified by the *ST_PrivateUnits* and *ST_PrivateOffsetUnits* attributes shall be a supported <unit name>. The value is a supported <unit name> if and only if the value is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.

15.1.2 ST_LRM Methods

Purpose

Return an ST_LRM value constructed from either:

a) the well-known text representation;

b) the GML representation;

c) the specified INTEGER lrmid, the CHARACTER VARYING LRM name, type and units, and the CHARACTER VARYING ARRAY constraint collection values; or

d) the specified INTEGER lrmid, the CHARACTER VARYING LRM name, type and units, the CHARACTER VARYING ARRAY constraint collection, the CHARACTER VARYING offset units and the positive lateral and vertical offset direction values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LRM
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LRM
   (anlrmid INTEGER,
    anlrmname CHARACTER VARYING(ST_MaxLRMNameLength),
    anlrmtype CHARACTER VARYING(128),
    aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength),
    aconstraintarray CHARACTER VARYING(ST_MaxConstraintLength)
      ARRAY[ST_MaxConstraintArrayElements])
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LRM
   (anlrmid INTEGER,
    anlrmname CHARACTER VARYING(ST_MaxLRMNameLength),
    anlrmtype CHARACTER VARYING(128),
    aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength),
    aconstraintarray CHARACTER VARYING(ST_MaxConstraintLength)
      ARRAY[ST_MaxConstraintArrayElements],
    anoffsetunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength),
    apositivelateraloffsetdirection CHARACTER VARYING(10),
    apositiveverticaloffsetdirection CHARACTER VARYING(10))
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxConstraintArrayElements* is the implementation-defined maximum cardinality of an array of CHARACTER VARYING constraint values.
2) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
3) *ST_MaxLRMNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a linear referencing method.
4) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
5) *ST_MaxConstraintLength* is the implementation-defined maximum length of the CHARACTER VARYING used for an LRM constraint.

Description

1) The method *ST_LRM(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) For the null-call type-preserving SQL-invoked constructor method *ST_LRM(CHARACTER LARGE OBJECT)*:

   Case:
   a) If *awktorgml* contains a LinearReferencingMethod XML element in the GML representation, then return the result of the value expression: *ST_LRMFromGML(awktorgml)*.
   b) Otherwise, return the result of the value expression: *ST_LRMFromText(awktorgml)*.
3) The method *ST_LRM(INTEGER, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING ARRAY)* takes the following input parameters:
   a) an INTEGER value *anlrmid,*
   b) a CHARACTER VARYING value *anlrmname,*
   c) a CHARACTER VARYING value *anlrmtype,*
   d) a CHARACTER VARYING value *aunitofmeasure,*
   e) a CHARACTER VARYING ARRAY value *aconstraintarray.*
4) For the type-preserving SQL-invoked constructor method *ST_LRM(INTEGER, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING ARRAY):*
   a) If *anlrmid* is the null value or if *anlrmname* is the null value or if *anlrmtype* is the null value or if *aunitofmeasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) The values for *aunitofmeasure* shall be a supported <unit name>.
   d) The value for *aunitofmeasure* is a supported <unit name> if and only if the value of *aunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.
   e) If the unit specified by *aunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
   f) Return an *ST_LRM* value with:
      i) The *ST_PrivateLRMID* attribute set to *anlrmid*.
      ii) The *ST_PrivateLRMName* attribute set to *anlrmname*.
      iii) The *ST_PrivateLRMType* attribute set to *anlrmtype*.
      iv) The *ST_PrivateUnits* attribute set to *aunitofmeasure*.

v) The *ST_PrivateConstraints* attribute set to *aconstraintarray*.

5) The method *ST_LRM(INTEGER, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING ARRAY, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

   a) an INTEGER value *anlrmid,*

   b) a CHARACTER VARYING value *anlrmname,*

   c) a CHARACTER VARYING value *anlrmtype,*

   d) a CHARACTER VARYING value *aunitofmeasure,*

   e) a CHARACTER VARYING ARRAY value *aconstraintarray,*

   f) a CHARACTER VARYING value *anoffsetunitofmeasure,*

   g) a CHARACTER VARYING value *apositivelateraloffsetdirection,*

   h) a CHARACTER VARYING value *apositiveverticaloffsetdirection.*

6) For the type-preserving SQL-invoked constructor method *ST_LRM(INTEGER, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING ARRAY, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING)*:

   a) If anlrmid is the null value or if anlrmname is the null value or if anlrmtype is the null value or if aunitofmeasure is the null value, then an exception condition is raised: SQL/MM Spatial exception – null argument.

   b) If SELF is the null value, then return the null value.

   c) The values for *aunitofmeasure* shall be a supported <unit name>.

   d) The value for *aunitofmeasure* is a supported <unit name> if and only if the value of *aunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.

   e) If the unit specified by *aunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

   f) If *anoffsetunitofmeasure* is NOT NULL, then:

      i) The values for *anoffsetunitofmeasure* shall be a supported <unit name>.

      ii) The value for *anoffsetunitofmeasure* is a supported <unit name> if and only if the value of *anoffsetunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.

      iii) If the unit specified by *anoffsetunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

   g) Return an *ST_LRM* value with:

      i) The *ST_PrivateLRMID* attribute set to *anlrmid*.

      ii) The *ST_PrivateLRMName* attribute set to *anlrmname*.

      iii) The *ST_PrivateLRMType* attribute set to *anlrmtype*.

      iv) The *ST_PrivateUnits* attribute set to *aunitofmeasure.*

      v) The *ST_PrivateConstraints* attribute set to *aconstraintarray*.

      vi) The *ST_PrivateOffsetUnits* attribute set to *anoffsetunitofmeasure*.

      vii) Case:

         1) If *anoffsetunitofmeasure* is NULL, then the *ST_PrivatePositiveLateralOffsetDirection* attribute set to NULL.

         2) Otherwise:

A) If *apositivelateraloffsetdirection* is NULL, then the *ST_PrivatePositiveLateralOffsetDirection* attribute set to ‘right’.

B) Otherwise, the *ST_PrivatePositiveLateralOffsetDirection* attribute set to *apositivelateraloffsetdirection*.

viii) Case:

1) If *anoffsetunitofmeasure* is NULL, then the *ST_PrivatePositiveVerticalOffsetDirection* attribute set to NULL.

2) Otherwise:

A) If *apositivelateraloffsetdirection* is NULL, then the *ST_PrivatePositiveVerticalOffsetDirection* attribute set to ‘up’.

B) Otherwise, the *ST_PrivatePositiveVerticalOffsetDirection* attribute set to *apositiveverticaloffsetdirection*.

15.1.3 ST_LRMID Methods

Purpose

Observe and mutate the attribute ST_PrivateLRMID of an ST_LRM value.

Definition

```
CREATE METHOD ST_LRMID()
   RETURNS INTEGER
   FOR ST_LRM
   RETURN SELF.ST_PrivateLRMID

CREATE METHOD ST_LRMID
   (anlrmid INTEGER)
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      IF anlrmid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRMID(anlrmid)
            END;
      END IF;
   END
```

Description

1) The method *ST_LRMID()* has no input parameters.

2) The null-call method *ST_LRMID()* returns the value of the *ST_PrivateLRMID* attribute.

3) The method *ST_LRMID(INTEGER)* takes the following input parameters:

   a) an INTEGER value *anlrmid*.

4) For the type-preserving method *ST_LRMID(INTEGER)*:

   Case:

   a) If *anlrmid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRM* value with the attribute *ST_PrivateLRMID* set to *anlrmid*.

15.1.4 ST_LRMName Methods

Purpose

Observe and mutate the attribute ST_PrivateLRMName of an ST_LRM value.

Definition

```
CREATE METHOD ST_LRMName()
   RETURNS CHARACTER VARYING(ST_MaxLRMNameLength)
   FOR ST_LRM
   RETURN SELF.ST_PrivateName

CREATE METHOD ST_LRMName
   (anlrmname CHARACTER VARYING(ST_MaxLRMNameLength))
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      IF anlrmname IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRMName(anlrmname)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxLRMNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a linear referencing method.

Description

1) The method *ST_LRMName()* has no input parameters.
2) The null-call method *ST_LRMName()* returns the value of the *ST_PrivateLRMName* attribute.
3) The method *ST_LRMName(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anlrmname*.
4) For the type-preserving method *ST_LRMName(CHARACTER VARYING)*:

   Case:

   a) If *anlrmname* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_LRM* value with the attribute *ST_PrivateLRMName* set to *anlrmname*.

15.1.5 ST_LRMType Methods

Purpose

Observe and mutate the attribute ST_PrivateLRMType of an ST_LRM value.

Definition

```
CREATE METHOD ST_LRMType()
   RETURNS CHARACTER VARYING(128)
   FOR ST_LRM
   RETURN SELF.ST_PrivateLRMType

CREATE METHOD ST_LRMType
   (anlrmtype CHARACTER VARYING(128))
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      IF anlrmtype IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRMType(anlrmtype)
            END;
      END IF;
   END
```

Description

1) The method *ST_LRMType()* has no input parameters.
2) The null-call method *ST_LRMType()* returns the value of the *ST_PrivateLRMType* attribute.
3) The method *ST_LRMType(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anlrmtype*.
4) For the type-preserving method *ST_LRMType(CHARACTER VARYING)*:

   Case:

   a) If *anlrmtype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_LRM* value with the attribute *ST_PrivateLRMType* set to *anlrmtype*.

15.1.6 ST_UnitOfMeasure Methods

Purpose

Observe and mutate the attribute ST_PrivateUnits of an ST_LRM value.

Definition

```
CREATE METHOD ST_UnitOfMeasure()
   RETURNS CHARACTER VARYING(ST_MaxUnitNameLength)
   FOR ST_LRM
   RETURN SELF.ST_PrivateUnits

CREATE METHOD ST_UnitOfMeasure
   (aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRM
   FOR ST_LRM
   BEGIN
      IF aunitofmeasure IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  --
                  -- See Description
                  --
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_UnitOfMeasure()* has no input parameters.
2) The null-call method *ST_UnitOfMeasure()* returns the value of the *ST_PrivateUnits* attribute.
3) The method *ST_UnitOfMeasure(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunitofmeasure*.
4) For the type-preserving method *ST_UnitOfMeasure(CHARACTER VARYING)*:

   Case:

   a) If *aunitofmeasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise:
      i) The values for *aunitofmeasure* shall be a supported <unit name>.
      ii) The value for *aunitofmeasure* is a supported <unit name> if and only if the value of *aunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.
      iii) If the unit specified by *aunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

iv) Return an *ST_LRM* value with the attribute *ST_PrivateUnits* set to *aunitofmeasure*.

15.1.7 ST_Constraints Methods

Purpose

Observe and mutate the attribute ST_PrivateConstraints of an ST_LRM value.

Definition

```
CREATE METHOD ST_Constraints()
   RETURNS CHARACTER VARYING(ST_MaxConstraintLength)
      ARRAY[ST_MaxConstraintArrayElements]
   FOR ST_LRM
   RETURN SELF.ST_PrivateConstraints

CREATE METHOD ST_Constraints
   (aconstraintarray CHARACTER VARYING(ST_MaxConstraintLength)
      ARRAY[ST_MaxConstraintArrayElements])
   RETURNS ST_LRM
   FOR ST_LRM
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateConstraints(aconstraintarray)
      END
```

Definitional Rules

1) *ST_MaxConstraintLength* is the implementation-defined maximum length of the CHARACTER VARYING used for an LRM constraint.

Description

1) The method *ST_Constraints()* has no input parameters.
2) The null-call method *ST_Constraints()* returns the value of the *ST_PrivateConstraints* attribute.
3) The method *ST_Constraints(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aconstraintarray*.
4) For the type-preserving method *ST_Constraints(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_LRM* value with the attribute *ST_PrivateConstraints* set to *aconstraintarray*.

15.1.8 ST_OffsetMeasUnit Methods

Purpose

Observe and mutate the attribute ST_PrivateOffsetUnits of an ST_LRM value.

Definition

```
CREATE METHOD ST_OffsetMeasUnit()
   RETURNS CHARACTER VARYING(ST_MaxUnitNameLength)
   FOR ST_LRM
   RETURN SELF.ST_PrivateOffsetUnits

CREATE METHOD ST_OffsetMeasUnit
   (anoffsetunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRM
   FOR ST_LRM
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            --
            -- See Description
            --
      END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_OffsetMeasUnit()* has no input parameters.
2) The null-call method *ST_OffsetMeasUnit()* returns the value of the *ST_PrivateOffsetUnits* attribute.
3) The method *ST_OffsetMeasUnit(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anoffsetunitofmeasure*.
4) For the type-preserving method *ST_OffsetMeasUnit(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise:
      i) The values for *anoffsetunitofmeasure* shall be a supported <unit name>.
      ii) The value for *anoffsetunitofmeasure* is a supported <unit name> if and only if the value of *anoffsetunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.
      iii) If the unit specified by *anoffsetunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
      iv) Let *PLOD* be the *SELF.ST_PrivatePositiveLateralOffsetDirection* attribute CHARACTER VARYING(10) value. Let *PVOD* be the *SELF.ST_PrivatePositiveVerticalOffsetDirection* attribute CHARACTER VARYING(10) value.
      v) If *anoffsetunitofmeasure* is NULL, then set *PLOD* = *PVOD* = NULL.
      vi) Otherwise:
         1) If *PLOD* is NULL, then set *PLOD* = ‘right’.
         2) If *PVOD* is NULL, then set *PVOD* = ‘up’.

vii) Return an *ST_LRM* value with the attribute *ST_PrivateOffsetUnits* set to *anoffsetunitofmeasure*, the ST_PrivatePositiveLateralOffsetDirection attribute set to *PLOD* and the ST_PrivatePositiveVerticalOffsetDirection attribute set to *PVOD*.

15.1.9 ST_PosLatOffsetDir Methods

Purpose

Observe and mutate the attribute ST_PrivatePositiveLateralOffsetDirection of an ST_LRM value.

Definition

```
CREATE METHOD ST_PosLatOffsetDir()
   RETURNS CHARACTER VARYING(10)
   FOR ST_LRM
   RETURN SELF.ST_PrivatePositiveLateralOffsetDirection

CREATE METHOD ST_PosLatOffsetDir
   (apositivelateraloffsetdirection CHARACTER VARYING(10))
   RETURNS ST_LRM
   FOR ST_LRM
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivatePositiveLateralOffsetDirection
             (apositivelateraloffsetdirection)
      END
```

Description

1) The method *ST_PosLatOffsetDir()* has no input parameters.
2) The null-call method *ST_PosLatOffsetDir()* returns the value of the *ST_PrivatePositiveLateralOffsetDirection* attribute.
3) The method *ST_PosLatOffsetDir(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *apositivelateraloffsetdirection*.
4) For the type-preserving method *ST_PosLatOffsetDir(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) If *apositivelateraloffsetdirection* is NOT NULL and *SELF.ST_PrivateOffsetUnits* is NULL, then an exception condition is raised: *SQL/MM Spatial exception – offset unit must be specified*.
   c) If *apositivelateraloffsetdirection* is NULL and SELF.*ST_PrivateOffsetUnits* is NOT NULL, return an *ST_LRM* value with the attribute *ST_PrivatePositiveLateralOffsetDirection* set to ‘right’.
   d) Otherwise, return an *ST_LRM* value with the attribute *ST_PrivatePositiveLateralOffsetDirection* set to *apositivelateraloffsetdirection*.

15.1.10 ST_PosVerOffsetDir Methods

Purpose

Observe and mutate the attribute ST_PrivatePositiveVerticalOffsetDirection of an ST_LRM value.

Definition

```
CREATE METHOD ST_PosVerOffsetDir()
   RETURNS CHARACTER VARYING(10)
   FOR ST_LRM
   RETURN SELF.ST_PrivatePositiveVerticalOffsetDirection

CREATE METHOD ST_PosVerOffsetDir
   (apositiveverticaloffsetdirection CHARACTER VARYING(10))
   RETURNS ST_LRM
   FOR ST_LRM
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivatePositiveVerticalOffsetDirection
             (apositiveverticaloffsetdirection)
      END
```

Description

1) The method *ST_PosVerOffsetDir()* has no input parameters.
2) The null-call method *ST_PosVerOffsetDir()* returns the value of the *ST_PrivatePositiveVerticalOffsetDirection* attribute.
3) The method *ST_PosVerOffsetDir(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *apositiveverticaloffsetdirection*.
4) For the type-preserving method *ST_PosVerOffsetDir(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) If apositiveverticaloffsetdirection is NOT NULL and SELF.ST_PrivateOffsetUnits is NULL, then an exception condition is raised: SQL/MM Spatial exception – offset unit must be specified.
   c) If apositiveverticaloffsetdirection is NULL and SELF.ST_PrivateOffsetUnits is NOT NULL, return an ST_LRM value with the attribute ST_PrivatePositiveLateralOffsetDirection set to ‘up’.
   d) Otherwise, return an ST_LRM value with the attribute ST_PrivatePositiveVerticalOffsetDirection set to apositiveverticaloffsetdirection.

15.1.11 ST_LRMFromText Function

Purpose

Return an ST_LRM value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRM value.

Definition

```
CREATE FUNCTION ST_LRMFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRM
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing type value.

Description

1) The function *ST_LRMFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awkt*.

2) For the null-call function *ST_LRMFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_LRM* value.

      If *awkt* is not producible in the BNF for <lrm text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_LRMFromText(awkt) AS ST_LRM)*.

15.1.12 ST_LRMFromGML Function

Purpose

Return an ST_LRM value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Linear Referencing Method representation of an ST_LRM value.

Definition

```
CREATE FUNCTION ST_LRMFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_LRM
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_LRMFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_LRMFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain a LinearReferencingMethod XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LRMFromGML(agml) AS ST_LRM)*.

15.2 ST_LinearElement Type and Routines

15.2.1 ST_LinearElement Type

Purpose

The ST_LinearElement type specifies the underlying linear element upon which the measures in the Linear Referencing System are made.

Definition

```
CREATE TYPE ST_LinearElement
   AS (
      ST_PrivateLinearElementID INTEGER DEFAULT NULL,
      ST_PrivateDefaultLRM INTEGER DEFAULT NULL,
      ST_PrivateDefaultMeasure ST_LRMeasure DEFAULT NULL,
      ST_PrivateLinearElementType CHARACTER VARYING(128) DEFAULT NULL,
      ST_PrivateStartValues ST_StartValue
         ARRAY[ST_MaxStartValueArrayElements] DEFAULT ARRAY[]
   )
   NOT INSTANTIABLE
   NOT FINAL

   METHOD ST_LinearElementID()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_LinearElementID
      (anleid INTEGER)
      RETURNS ST_LinearElement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_DefaultLRM()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_DefaultLRM
      (anlrmid INTEGER)
      RETURNS ST_LinearElement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_DefaultMeasure()
      RETURNS ST_LRMeasure
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_DefaultMeasure
   (ameasure ST_LRMeasure)
   RETURNS ST_LinearElement
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LEType()
   RETURNS CHARACTER VARYING(128)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LEType
   (alinearelementtype CHARACTER VARYING(128))
   RETURNS ST_LinearElement
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartValue
   (anlrmid INTEGER)
   RETURNS ST_LRMeasure
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_StartValue
   (anlrmid INTEGER,
    ameasure ST_LRMeasure)
   RETURNS ST_LinearElement
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_TranslateToInst
   (asourcepositionexpression ST_PositionExp,
    atargetlinearelement ST_LinearElement,
    atargetLRM INTEGER)
   RETURNS ST_DistanceExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_TranslateToType
   (asourcepositionexpression ST_PositionExp,
    atargetlinearelementtype CHARACTER VARYING(128),
    atargetLRM INTEGER)
   RETURNS ST_PositionExp ARRAY[ST_MaxPositionExpArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxPositionExpArrayElements* is the implementation-defined maximum cardinality of an array of *ST_PositionExp* values.
2) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.
3) The attribute *ST_PrivateLinearElementID* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLinearElementID*.
4) The attribute *ST_PrivateDefaultLRM* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateDefaultLRM*.
5) The attribute *ST_PrivateDefaultMeasure* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateDefaultMeasure*.
6) The attribute *ST_PrivateLinearElementType* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLinearElementType*.
7) The attribute *ST_PrivateStartValues* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateStartValues*.

Description

1) The *ST_LinearElement* type provides for public use:
   a) a method *ST_LinearElementID()*,
   b) a method *ST_LinearElementID(INTEGER)*,
   c) a method *ST_DefaultLRM()*,
   d) a method *ST_DefaultLRM(INTEGER)*,
   e) a method *ST_DefaultMeasure()*,
   f) a method *ST_DefaultMeasure(ST_LRMeasure)*,
   g) a method *ST_LEType()*,
   h) a method *ST_LEType(CHARACTER VARYING)*,
   i) a method *ST_StartValue(INTEGER)*,
   j) a method *ST_StartValue(INTEGER, ST_LRMeasure)*,
   k) a method *ST_TranslateToInst(ST_PositionExp, ST_LinearElement, INTEGER)*,
   l) a method *ST_TranslateToType(ST_PositionExp, CHARACTER VARYING, INTEGER)*,
   m) a function *ST_LEFromText(CHARACTER LARGE OBJECT)*,
   n) a function *ST_LEFeatFromGML(CHARACTER LARGE OBJECT).*
2) The *ST_PrivateLinearElementID* attribute contains the INTEGER linear element ID leid value.
3) The *ST_PrivateDefaultLRM* attribute contains the INTEGER default LRM lrmid value.
4) The *ST_PrivateDefaultMeasure* attribute contains the *ST_LRMeasure* default measure value.
5) The *ST_PrivateLinearElementType* attribute optionally contains the CHARACTER VARYING linear element type value.

6) The *ST_PrivateStartValues* attribute optionally contains the collection of *ST_StartValue* values.

7) The *ST_PrivateDefaultLRM* attribute value is the lrmid of the LRM used for all measurements made along the *ST_LinearElement* unless specified otherwise in an *ST_PositionExp* or otherwise explicitly overridden.

8) The *ST_PrivateDefaultMeasure* attribute value is the default length (or weight) value used in all calculations requiring a total length for the linear element (e.g., interpolative LRM calculations). Changing this value would impact previous as well as future linearly referenced location values.

9) The *ST_PrivateUnits* attribute of the *ST_LRMeasure* value specified by the *ST_PrivateDefaultMeasure* attribute shall not be NULL.

10) The *ST_PrivateLinearElementType* attribute value is a user-definable type for the linear element used by the *ST_TranslateToType* method to restrict the set of candidate target linear elements.

11) For each of the *ST_StartValue* values specified by the *ST_PrivateStartValues* attribute, if the *ST_PrivateUnits* attribute of the *ST_LRMeasure* value specified by the *ST_PrivateMeasure* attribute of the *ST_StartValue* value is NULL, then the *ST_PrivateUnits* attribute value of the *ST_LRM* value specified by the *ST_PrivateLRM* attribute of the *ST_StartValue* value shall apply.

12) The translation between position expressions having different linear elements and/or LRMs must be closed (source and target must be position expressions), commutative (translating the target back to the source shall result in the original source location) and transitive (for all A, B and C position expressions, translating from A to B and then to C should be equivalent to translating from A directly to C).

13) Position expression translation is dependent upon the mappings defined between linear elements and LRMs. It is implementation-defined how these mappings are defined and how the system chooses which one(s) to use for a given position expression translation.

14) For the method *ST_TranslateToInst*, if the source and target linear elements are collinear or intersecting at the location specified by the source position expression, then the returned linearly referenced location shall be spatially equal to the linearly referended location specified by the source *ST_PositionExp*. Otherwise, it shall be implementation-defined whether the translation should follow a normal from the source or the target in order to insure that the commutative relationship holds.

15) For the method *ST_TranslateToType*, the target Linear Referencing Method defaults to the defaultLRM of the target linear element type, unless explicitly overridden with the "targetLRM" input parameter. This operation is used instead of *ST_TranslateToInst* when the source has been mapped to a set of contiguous target instances and it is not known a priori on which instances the position will fall upon.

15.2.2 ST_LinearElementID Methods

Purpose

Observe and mutate the attribute ST_PrivateLinearElementID of an ST_LinearElement value.

Definition

```
CREATE METHOD ST_LinearElementID()
   RETURNS INTEGER
   FOR ST_LinearElement
   RETURN SELF.ST_PrivateLinearElementID

CREATE METHOD ST_LinearElementID
   (anleid INTEGER)
   RETURNS ST_LinearElement
   FOR ST_LinearElement
   BEGIN
      IF anleid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLinearElementID(anleid)
            END;
      END IF;
   END
```

Description

1) The method *ST_LinearElementID()* has no input parameters.
2) The null-call method *ST_LinearElementID()* returns the value of the *ST_PrivateLinearElementID* attribute.
3) The method *ST_LinearElementID(INTEGER)* takes the following input parameters:
   a) an INTEGER value *anleid*.
4) For the type-preserving method *ST_LinearElementID(INTEGER)*:
   a) If *anleid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_LinearElement* value with the attribute *ST_PrivateLinearElementID* set to *anleid*.

15.2.3 ST_DefaultLRM Methods

Purpose

Observe and mutate the attribute ST_PrivateDefaultLRM of an ST_LinearElement value.

Definition

```
CREATE METHOD ST_DefaultLRM()
   RETURNS INTEGER
   FOR ST_LinearElement
   RETURN SELF.ST_PrivateDefaultLRM

CREATE METHOD ST_DefaultLRM
   (anlrmid INTEGER)
   RETURNS ST_LinearElement
   FOR ST_LinearElement
   BEGIN
      IF anlrmid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateDefaultLRM(anlrmid)
            END;
      END IF;
   END
```

Description

1) The method *ST_DefaultLRM()* has no input parameters.

2) The null-call method *ST_DefaultLRM()* returns the value of the *ST_PrivateDefaultLRM* attribute.

3) The method *ST_DefaultLRM(INTEGER)* takes the following input parameters:

   a) an INTEGER value *anlrmid*.

4) For the type-preserving method *ST_DefaultLRM(INTEGER)*:

   a) If *anlrmid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LinearElement* value with the attribute *ST_PrivateDefaultLRM* set to *anlrmid*.

15.2.4 ST_DefaultMeasure Methods

Purpose

Observe and mutate the attribute ST_PrivateDefaultMeasure of an ST_LinearElement value.

Definition

```
CREATE METHOD ST_DefaultMeasure()
   RETURNS ST_LRMeasure
   FOR ST_LinearElement
   RETURN SELF.ST_PrivateDefaultMeasure

CREATE METHOD ST_DefaultMeasure
   (ameasure ST_LRMeasure)
   RETURNS ST_LinearElement
   FOR ST_LinearElement
   BEGIN
      IF ameasure IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         CASE
            WHEN SELF IS NULL THEN RETURN NULL
            ELSE
               BEGIN
                  SIGNAL SQLSTATE '01F82'
                     SET MESSAGE_TEXT = 'changing default measure may
                      invalidate position expressions using this linear
                      element';
                  RETURN SELF.ST_PrivateDefaultMeasure(ameasure);
               END
         END;
      END IF;
   END
```

Description

1) The method *ST_DefaultMeasure()* has no input parameters.

2) The null-call method *ST_DefaultMeasure()* returns the value of the *ST_PrivateDefaultMeasure* attribute.

3) The method *ST_DefaultMeasure(ST_LRMeasure)* takes the following input parameters:

   a) an *ST_LRMeasure* value *ameasure*.

4) For the type-preserving method *ST_DefaultMeasure(ST_LRMeasure)*:

   a) If *ameasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise,

      i) A completion condition is raised: *SQL/MM Spatial warning – changing default measure may invalidate position expressions using this linear element.*

      ii) Return an *ST_LinearElement* value with the attribute *ST_PrivateDefaultMeasure* set to *ameasure*.

15.2.5 ST_LEType Methods

Purpose

Observe and mutate the attribute ST_PrivateLinearElementType of an ST_LinearElement value.

Definition

```
CREATE METHOD ST_LEType()
   RETURNS CHARACTER VARYING(128)
   FOR ST_LinearElement
   RETURN SELF.ST_PrivateLinearElementType

CREATE METHOD ST_LEType
   (alinearelementtype CHARACTER VARYING(128))
   RETURNS ST_LinearElement
   FOR ST_LinearElement
   BEGIN
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               SELF.ST_PrivateLinearElementType(alinearelementtype)
         END;
   END
```

Description

1) The method *ST_LEType()* has no input parameters.

2) The null-call method *ST_LEType()* returns the value of the *ST_PrivateLinearElementType* attribute.

3) The method *ST_LEType(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *alinearelementtype*.

4) For the type-preserving method *ST_LEType(CHARACTER VARYING)*:

   a) If SELF is the null value, then return the null value.

   b) Otherwise, return an *ST_LinearElement* value with the attribute *ST_PrivateLinearElementType* set to *alinearelementtype*.

15.2.6 ST_StartValue Methods

Purpose

Observe and mutate the measure value at the start of the ST_LinearElement for the specified Linear Referencing Method.

Definition

```
CREATE METHOD ST_StartValue
   (anlrmid INTEGER)
   RETURNS ST_LRMeasure
   FOR ST_LinearElement
   BEGIN
      IF anlrmid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      DECLARE counter INTEGER;
      DECLARE measure ST_LRMeasure;
      DECLARE units CHARACTER VARYING;
      DECLARE found BOOLEAN;
      DECLARE lrm ST_LRM;
      -- Find the ST_LRM value having an ST_PrivateLRMID = anlrmid
         -- If found, set lrm = the found ST_LRM value
         -- else SIGNAL SQLSTATE '2FF81'
         --  SET MESSAGE_TEXT = 'invalid LRM'
            --
            -- See Description
            --
      -- Get units for the input LRM
      SET units = lrm.ST_UnitOfMeasure();
      -- Set measure to be the default (zero) value
      SET measure = NEW ST_LRMeasure(0, units)
      -- Search for a Start Value to override the default
      SET counter = 1;
      SET found = FALSE;
      WHILE counter <= CARDINALITY(SELF.ST_PrivateStartValues)
       AND NOT found DO
         -- If a start value for the input LRM is found,
         -- retrieve the start value's measure value
         IF anlrmid = SELF.ST_PrivateStartValues[counter].ST_LRM() THEN
            -- override the default measure value
            BEGIN
               SET measure =
                SELF.ST_PrivateStartValues[counter].ST_LRMeasure();
               SET found = TRUE;
            END
         END IF;
      END WHILE;
      RETURN measure;
   END

CREATE METHOD ST_StartValue
   (anlrmid INTEGER,
    ameasure ST_LRMeasure)
   RETURNS ST_LinearElement
   FOR ST_LinearElement
   BEGIN
      IF anlrmid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
```

```
      ELSEIF ameasure IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      DECLARE counter INTEGER;
      DECLARE startvaluearray ST_StartValue
         ARRAY[ST_MaxStartValueArrayElements];
      DECLARE newstartvalue ST_StartValue;
      -- Search for a Start Value having the input LRM value
      SET counter = 1;
      WHILE counter <= CARDINALITY(SELF.ST_PrivateStartValues) DO
         -- If LRM is found, update its measure
         IF anlrmid = SELF.ST_PrivateStartValues[counter].ST_LRM() THEN
            BEGIN
               SET SELF.ST_PrivateStartValues[counter].ST_LRMeasure() =
                ameasure;
               -- return the ST_LinearElement with the new measure value
               RETURN SELF.ST_PrivateStartValues[counter];
            END
         END IF;
      END WHILE;
      -- No Start Value having the input LRM value; add one
      SET startvaluearray = SELF.ST_PrivateStartValues;
      SET newstartvalue = NEW ST_StartValue(anlrmid, ameasure);
      RETURN SELF.ST_PrivateStartValues(startvaluearray || newstartvalue);
   END
```

Description

1) The method *ST_StartValue(INTEGER)* takes the following input parameters:

   a) an INTEGER value *anlrmid*.

2) For the method *ST_StartValue(INTEGER):*

   a) If *anlrmid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If there does not exist an *ST_LRM* value having an *ST_PrivateLRMID = anlrmid*, then an exception condition is raised: *SQL/MM Spatial exception – invalidLRM*.

   c) If the *ST_PrivateStartValues* attribute contains an *ST_StartValue* having an INTEGER *ST_PrivateLRM* attribute value equal to *anlrmid*, return the corresponding *ST_PrivateMeasure* attribute *ST_LRMeasure* value of that *ST_StartValue*.

   d) Otherwise,

      i) Let *L* be the *ST_LRM* value such that *L.ST_LRMID = anlrmid*.

      ii) Return an *ST_LRMeasure* value having an *ST_PrivateMeasure* value equal to 0 (zero) and an *ST_PrivateUnits* value equal to *L.ST_UnitOfMeasure()*.

3) The method *ST_StartValue(INTEGER, ST_LRMeasure)* takes the following input parameters:

   a) an INTEGER value *anlrmid*,

   b) an *ST_LRMeasure* value *ameasure*.

4) For the type-preserving method *ST_StartValue(INTEGER, ST_LEMeasure)*:

   a) If *anlrmid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *ameasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If the *ST_PrivateStartValues* attribute contains an *ST_StartValue* having an *ST_PrivateLRM* attribute value equal to *anlrmid*, return an *ST_LinearElement* value with the *ST_PrivateMeasure* attribute *ST_LRMeasure* value of that *ST_StartValue* set to *ameasure*.

d) Otherwise,

   i) Construct a new *ST_StartValue* having an *ST_PrivateLRM* attribute value equal to *anlrmid* and an *ST_PrivateMeasure* attribute *ST_LRMeasure* value equal to *ameasure*.

   ii) Return an *ST_LinearElement* with an *ST_PrivateStartValues* attribute set to the concatenation of its original *ST_StartValue* ARRAY value with the new *ST_StartValue*.

15.2.7 ST_TranslateToInst Method

Purpose

Translate an ST_PositionExp defined along the subject (source) ST_LinearElement into an ST_DistanceExp measured along a known, specified target ST_LinearElement using the target Linear Referencing Method.

Definition

```
CREATE METHOD ST_TranslateToInst
   (asourcepositionexpression ST_PositionExp,
    atargetlinearelement ST_LinearElement,
    atargetLRM INTEGER)
   RETURNS ST_DistanceExp
   FOR ST_LinearElement
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_TranslateToInst(ST_PositionExp*, *ST_LinearElement, INTEGER)* takes the following input parameters:
   a) an *ST_PositionExp* value *asourcepositionexpression*.
   b) an *ST_LinearElement* value *atargetlinearelement*.
   c) an INTEGER value *atargetLRM*.
2) For the type-preserving method *ST_TranslateToInst(ST_PositionExp*, *ST_LinearElement, INTEGER)*:
   a) If *asourcepositionexpression* or *atargetlinearelement* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atargetLRM* is the null value, then *atargetLRM* is set to the value of the *ST_PrivateDefaultLRM* attribute of *atargetlinearelement.*
   c) If it is not possible to perform the translation, then an exception condition is raised: *SQL/MM Spatial exception – cannot translate.*
   d) Return an *ST_DistanceExp* value that, when measured along *atargetlinearelement* using *atargetLRM*, specifies a linearly referenced location that is equivalent to the one specified by *asourcepositionexpression*.

15.2.8 ST_TranslateToType Method

Purpose

Translate an ST_PositionExp defined along the subject (source) ST_LinearElement into one or more ST_PositionExps measured along the appropriate instances of the element type specified, using the target Linear Referencing Method.

Definition

```
CREATE METHOD ST_TranslateToType
   (asourcepositionexpression ST_PositionExp,
    atargetlinearelementtype CHARACTER VARYING(256),
    atargetLRM INTEGER)
   RETURNS ST_PositionExp ARRAY[ST_MaxPositionExpArrayElements]
   FOR ST_LinearElement
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxPositionExpArrayElements* is the implementation-defined maximum cardinality of an array of *ST_PositionExp* values.

Description

1) The method *ST_TranslateToType(ST_PositionExp*, *CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) an *ST_PositionExp* value *asourcepositionexpression*.
   b) a CHARACTER VARYING value *atargetlinearelementtype*.
   c) an INTEGER value *atargetLRM*.
2) For the type-preserving method *ST_TranslateToType(ST_PositionExp*, *CHARACTER VARYING, INTEGER)*:
   a) If *asourcepositionexpression* or *atargetlinearelementtype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *atargetLRM* is the null value, then *atargetLRM* is set to the value of the *ST_PrivateDefaultLRM* attribute of the appropriate instance of the *atargetlinearelementtype. b+1)If it is not possible to perform the translation, then an exception condition is raised: SQL/MM Spatial exception – cannot translate.*
   c) If it is not possible to perform the translation, then an exception condition is raised: *SQL/MM Spatial exception – cannot translate.*
   d) Return the *ST_PositionExp* values which represent linearly referenced locations along *ST_LinearElements* having an *ST_PrivateLinearElementType* equal to *atargetlinearelementtype* and equivalent to the linearly referenced location specified by *asourcepositionexpression*.

15.2.9 ST_LEFromText Function

Purpose

Return an ST_LinearElement value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LinearElement value.

Definition

```
CREATE FUNCTION ST_LEFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LinearElement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing feature value.

Description

1) The function *ST_LEFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_LEFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_LinearElement* value.

      If *awkt* is not producible in the BNF for <linear element text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_LEFromText(awkt) AS ST_LinearElement)*.

15.2.10 ST_LEFromGML Function

Purpose

Return an ST_LinearElement value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LinearElement value.

Definition

```
CREATE FUNCTION ST_LEFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_LinearElement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_LEFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_LEFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain a LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LEFromGML(agml) AS ST_LinearElement)*.

15.3 ST_LRFeature Type and Routines

15.3.1 ST_LRFeature Type

Purpose

The ST_LRFeature subtype of ST_LinearElement specifies any feature which can be linearly measured, that is, which supports the methods of ST_LinearElement. The ST_LRFeature type is instantiable.

Definition

```
CREATE TYPE ST_LRFeature
   UNDER ST_LinearElement
   AS (
      ST_PrivateFeatureID CHARACTER VARYING(ST_MaxFeatureIDLength)
         DEFAULT NULL,
      ST_PrivateReferents ST_Referent ARRAY[ST_MaxReferentArrayElements]
         DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LRFeature
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
      RETURNS ST_LRFeature
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LRFeature
      (anleid INTEGER,
       anlrmid INTEGER,
       ameasure ST_LRMeasure,
       alinearelementtype CHARACTER VARYING(128),
       astartvaluearray ST_StartValue
          ARRAY[ST_MaxStartValueArrayElements],
       afeatureID CHARACTER VARYING(ST_MaxFeatureIDLength))
      RETURNS ST_LRFeature
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
```



```
CONSTRUCTOR METHOD ST_LRFeature
   (anleid INTEGER,
    anlrmid INTEGER,
    ameasure ST_LRMeasure,
    alinearelementtype CHARACTER VARYING(128),
    astartvaluearray ST_StartValue
       ARRAY[ST_MaxStartValueArrayElements],
    afeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    areferentarray ST_Referent ARRAY[ST_MaxReferentArrayElements])
   RETURNS ST_LRFeature
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_FeatureID()
   RETURNS ST_LinearElement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_FeatureID
   (afeatureID CHARACTER VARYING(ST_MaxFeatureIDLength))
   RETURNS ST_LRFeature
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Referents()
   RETURNS ST_Referent ARRAY[ST_MaxReferentArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Referents
   (areferentarray ST_Referent ARRAY[ST_MaxReferentArrayElements])
   RETURNS ST_LRFeature
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.

2) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.

3) *ST_MaxReferentArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Referent* values.

4) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

5) The attribute *ST_PrivateFeatureID is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateFeatureID.*

6) The attribute *ST_PrivateReferents is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferents.*

Description

1) The *ST_LRFeature* type provides for public use:
   a) a method *ST_LRFeature(CHARACTER LARGE OBJECT)*,
   b) a method *ST_LRFeature(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER VARYING)*,
   c) a method *ST_LRFeature(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER VARYING, ST_Referent ARRAY)*,
   d) a method *ST_FeatureID()*,
   e) a method *ST_FeatureID(CHARACTER VARYING)*,
   f) a method *ST_Referents()*,
   g) a method *ST_Referents(ST_Referent ARRAY)*,
   h) a function *ST_LRFeatFromText(CHARACTER LARGE OBJECT)*,
   i) a function *ST_LRFeatFromGML(CHARACTER LARGE OBJECT).*
2) The *ST_PrivateFeatureID* attribute contains the CHARACTER VARYING feature ID value.
3) The *ST_PrivateReferents* attribute optionally contains the *ST_Referent* ARRAY collection of referent values.

15.3.2 ST_LRFeature Methods

Purpose

Return an ST_LRFeature value constructed from either:

a) the well-known text representation;

b) GML representation;

c) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, *ST_StartValue* ARRAY and CHARACTER VARYING feature ID values;

d) the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, *ST_StartValue* ARRAY, CHARACTER VARYING feature ID and ST_Referents ARRAY referent values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LRFeature
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRFeature
   FOR ST_LRFeature
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LRFeature
   (anleid INTEGER,
    anlrmid INTEGER,
    ameasure ST_LRMeasure,
    alinearelementtype CHARACTER VARYING(128),
    astartvaluearray ST_StartValue
       ARRAY[ST_MaxStartValueArrayElements],
    afeatureID CHARACTER VARYING(ST_MaxFeatureIDLength))
   RETURNS ST_LRFeature
   FOR ST_LRFeature
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LRFeature
   (anleid INTEGER,
    anlrmid INTEGER,
    ameasure ST_LRMeasure,
    alinearelementtype CHARACTER VARYING(128),
    astartvaluearray ST_StartValue
      ARRAY[ST_MaxStartValueArrayElements],
    afeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    areferentarray ST_Referent ARRAY[ST_MaxReferentArrayElements])
   RETURNS ST_LRFeature
   FOR ST_LRFeature
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.

2) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.

3) *ST_MaxReferentArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Referent* values.

4) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

Description

1) The method *ST_LRFeature(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) For the null-call type-preserving SQL-invoked constructor method *ST_LRFeature(CHARACTER LARGE OBJECT)*:

   Case:

   a) If *awktorgml* contains a feature type of LinearElement XML element in the GML representation, then return the result of the value expression: *ST_LRFeatFromGML(awktorgml)*.

   b) Otherwise, return the result of the value expression: *ST_LRFeatFromText(awktorgml)*.

3) The method *ST_LRFeature(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER VARYING)* takes the following input parameters:
   a) an INTEGER value *anleid,*
   b) an INTEGER value *anlrmid,*
   c) an *ST_LRMeasure* value *ameasure,*
   d) a CHARACTER VARYING value *alinearelementtype,*
   e) an *ST_StartValue* ARRAY value *astartvaluearray,*
   f) a CHARACTER VARYING value *afeatureID.*

4) The type-preserving SQL-invoked constructor method *ST_LRFeature(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER VARYING)*:

   Case:

   a) If *anleid* is the null value or if *anlrmid* is the null value or if *ameasure* is the null value or if *afeatureID* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRFeature* value with:
      i) The *ST_PrivateLinearElementID* attribute set to *anleid.*
      ii) The *ST_PrivateDefaultLRM* attribute set to *anlrmid.*
      iii) The *ST_PrivateDefaultMeasure* attribute set to *ameasure.*
      iv) The *ST_PrivateLinearElementType* attribute set to *alinearelementtype.*
      v) The *ST_PrivateStartValues* attribute set to *astartvaluearray.*
      vi) The *ST_PrivateFeatureID* attribute set to *afeatureID.*

5) The method *ST_LRFeature(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER VARYING, ST_Referent ARRAY)* takes the following input parameters:
   a) an INTEGER value *anleid,*

b) an INTEGER value *anlrmid,*

c) an *ST_LRMeasure* value *ameasure,*

d) a CHARACTER VARYING value *alinearelementtype,*

e) an *ST_StartValue* ARRAY value *astartvaluearray,*

f) a CHARACTER VARYING value *afeatureID,*

g) an *ST_Referent* ARRAY value *areferentarray.*

6) The type-preserving SQL-invoked constructor method *ST_LRFeature(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER VARYING, ST_Referent ARRAY)*:

Case:

a) If *anleid* is the null value or if *anlrmid* is the null value or if *ameasure* is the null value or if *afeatureID* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_LRFeature* value with:

i) The *ST_PrivateLinearElementID* attribute set to *anleid.*

ii) The *ST_PrivateDefaultLRM* attribute set to *anlrmid.*

iii) The *ST_PrivateDefaultMeasure* attribute set to *ameasure.*

iv) The *ST_PrivateLinearElementType* attribute set to *alinearelementtype.*

v) The *ST_PrivateStartValues* attribute set to *astartvaluearray.*

vi) The *ST_PrivateFeatureID* attribute set to *afeatureID.*

vii) The *ST_PrivateReferents* attribute set to *areferentarray.*

15.3.3 ST_FeatureID Methods

Purpose

Observe and mutate the attribute ST_PrivateFeatureID of an ST_LRFeature value.

Definition

```
CREATE METHOD ST_FeatureID()
   RETURNS CHARACTER VARYING(ST_MaxFeatureIDLength)
   FOR ST_LRFeature
   RETURN SELF.ST_PrivateFeatureID

CREATE METHOD ST_FeatureID
   (afeatureID CHARACTER VARYING(ST_MaxFeatureIDLength))
   RETURNS ST_LRFeature
   FOR ST_LRFeature
   BEGIN
      IF afeatureID IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateFeatureID(afeatureID)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

Description

1) The method *ST_FeatureID()* has no input parameters.

2) The null-call method *ST_FeatureID()* returns the value of the *ST_PrivateFeatureID* attribute.

3) The method *ST_FeatureID(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *afeatureID*.

4) For the type-preserving method *ST_FeatureID(CHARACTER VARYING)*:

   Case:

   a) If *afeatureID* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRFeature* value with the attribute *ST_PrivateFeatureID* set to *afeatureID*.

15.3.4 ST_Referents Methods

Purpose

Observe and mutate the attribute ST_PrivateReferents of an ST_LRFeature value.

Definition

```
CREATE METHOD ST_Referents()
   RETURNS ST_Referent ARRAY[ST_MaxReferentArrayElements]
   FOR ST_LRFeature
   RETURN SELF.ST_PrivateReferents

CREATE METHOD ST_Referents
   (areferentarray ST_Referent ARRAY[ST_MaxReferentArrayElements])
   RETURNS ST_LRFeature
   FOR ST_LRFeature
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateReferents(areferentarray)
      END
```

Definitional Rules

1) *ST_MaxReferentArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Referent* values.

Description

1) The method *ST_Referents()* has no input parameters.
2) The null-call method *ST_Referents()* returns the value of the *ST_PrivateReferents* attribute.
3) The method *ST_Referents(ST_Referent ARRAY)* takes the following input parameters:
   a) an *ST_Referent* ARRAY value *areferentarray.*
4) For the type-preserving method *ST_Referents(ST_Referent ARRAY)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_LRFeature* value with the attribute *ST_PrivateReferents* set to *areferentarray.*

15.3.5 ST_LRFeatFromText Function

Purpose

Return an ST_LRFeature value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRFeature value.

Definition

```
CREATE FUNCTION ST_LRFeatFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRFeature
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing feature value.

Description

1) The function *ST_LRFeatFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_LRFeatFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_LRFeature* value.

      If *awkt* is not producible in the BNF for <lr feature text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LRFeatFromText(awkt) AS ST_LRFeature)*.

15.3.6 ST_LRFeatFromGML Function

Purpose

Return an ST_LRFeature value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRFeature value.

Definition

```
CREATE FUNCTION ST_LRFeatFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_LRFeature
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_LRFeatFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_LRFeatFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain a feature type of LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LRFeatFromGML(agml) AS ST_LRFeature)*.

15.4 ST_LRCurve Type and Routines

15.4.1 ST_LRCurve Type

Purpose

The ST_LRCurve subtype of ST_LinearElement specifies any one-dimensional geometry of type ST_Curve which can be linearly measured, that is, which supports the methods of ST_LinearElement. The ST_LRCurve type is instantiable.

Definition

```
CREATE TYPE ST_LRCurve
   UNDER ST_LinearElement
   AS (
      ST_PrivateCurve ST_Curve DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LRCurve
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
      RETURNS ST_LRCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LRCurve
      (anleid INTEGER,
       anlrmid INTEGER,
       ameasure ST_LRMeasure,
       alinearelementtype CHARACTER VARYING(128),
       astartvaluearray ST_StartValue
          ARRAY[ST_MaxStartValueArrayElements],
       acurve ST_Curve)
      RETURNS ST_LRCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_Curve()
      RETURNS ST_Curve
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Curve
      (acurve ST_Curve)
      RETURNS ST_LRCurve
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
```

```
METHOD ST_Point
   (apositionexpression ST_PositionExp)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LRPosition
   (apoint ST_Point)
   RETURNS ST_PositionExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
2) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.
3) The attribute *ST_PrivateCurve is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateCurve*.

Description

1) The *ST_LRCurve* type provides for public use:
   a) a method *ST_LRCurve(CHARACTER LARGE OBJECT)*,
   b) a method *ST_LRCurve(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, ST_Curve)*,
   c) a method *ST_Curve()*,
   d) a method *ST_Curve(ST_Curve)*,
   e) a method *ST_Point(ST_PositionExp)*,
   f) a method *ST_LRPosition(ST_Point)*,
   g) a function *ST_LRFeatFromText(CHARACTER LARGE OBJECT)*,
   h) a function *ST_LRFeatFromGML(CHARACTER LARGE OBJECT).*
2) The *ST_PrivateCurve* attribute contains the *ST_Curve* curve value.

15.4.2 ST_LRCurve Methods

Purpose

Return an ST_LRCurve value constructed from either the well-known text representation; the GML representation; or the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY and ST_Curve curve values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LRCurve
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRCurve
   FOR ST_LRCurve
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LRCurve
   (anleid INTEGER,
    anlrmid INTEGER,
    ameasure ST_LRMeasure,
    alinearelementtype CHARACTER VARYING(128),
    astartvaluearray ST_StartValue
      ARRAY[ST_MaxStartValueArrayElements],
    acurve ST_Curve)
   RETURNS ST_LRCurve
   FOR ST_LRCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.

2) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.

Description

1) The method *ST_LRCurve(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) For the null-call type-preserving SQL-invoked constructor method *ST_LRCurve(CHARACTER LARGE OBJECT)*:

   Case:

   a) If *awktorgml* contains a curve type of LinearElement XML element in the GML representation, then return the result of the value expression: *ST_LRCurveFromGML(awktorgml)*.

   b) Otherwise, return the result of the value expression: *ST_LRCurveFromText(awktorgml)*.

3) The method *ST_LRCurve(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, ST_Curve)* takes the following input parameters:

   a) an INTEGER value *anleid,*

   b) an INTEGER value *anlrmid,*

   c) an *ST_LRMeasure* value *ameasure,*

d) a CHARACTER VARYING value *alinearelementtype,*

e) an *ST_StartValue* ARRAY value *astartvaluearray,*

f) an *ST_Curve* value *acurve.*

4) The type-preserving SQL-invoked constructor method *ST_LRCurve(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, ST_Curve)*:

Case:

a) If *anleid* is the null value or if *anlrmid* is the null value or if *ameasure* is the null value or if *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_LRCurve* value with:

i) The *ST_PrivateLinearElementID* attribute set to *anleid.*

ii) The *ST_PrivateDefaultLRM* attribute set to *anlrmid.*

iii) The *ST_PrivateDefaultMeasure* attribute set to *ameasure.*

iv) The *ST_PrivateLinearElementType* attribute set to *alinearelementtype.*

v) The *ST_PrivateStartValues* attribute set to *astartvaluearray.*

vi) The *ST_PrivateCurve* attribute set to *acurve.*

15.4.3 ST_Curve Methods

Purpose

Observe and mutate the attribute ST_PrivateCurve of an ST_LRCurve value.

Definition

```
CREATE METHOD ST_Curve()
   RETURNS ST_Curve
   FOR ST_LRCurve
   RETURN SELF.ST_PrivateCurve

CREATE METHOD ST_Curve
   (acurve ST_Curve)
   RETURNS ST_LRCurve
   FOR ST_LRCurve
   BEGIN
      IF acurve IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateCurve(acurve)
            END;
      END IF;
   END
```

Description

1) The method *ST_Curve()* has no input parameters.
2) The null-call method *ST_Curve()* returns the value of the *ST_PrivateCurve* attribute.
3) The method *ST_Curve(ST_Curve)* takes the following input parameters:
   a) an *ST_Curve* value *acurve*.
4) For the type-preserving method *ST_Curve(ST_Curve)*:

   Case:

   a) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_LRCurve* value with the attribute *ST_PrivateCurve* set to *acurve*.

15.4.4 ST_Point Method

Purpose

Return an ST_Point value representing the spatial position spatially equal to the linearly referenced location specified by an ST_PositionExp having an ST_LRCurve subtype of ST_LinearElement.

Definition

```
CREATE METHOD ST_Point
   (apositionexpression ST_PositionExp)
   RETURNS ST_Point
   FOR ST_LRCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Point(ST_PositionExp)* takes the following input parameters:
   a) an *ST_PositionExp* value *apositionexpression*.
2) For the type-preserving method *ST_Point(ST_PositionExp)*:
   a) If *apositionexpression* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) Return an *ST_Point* value representing the spatial position spatially equal to the linearly referenced location specified by *apositionexpression*.

15.4.5 ST_LRPosition Method

Purpose

Determine the linearly referenced location of a point on the ST_LinearElement of type ST_LRCurve closest to the given ST_Point value using the default Linear Referencing Method of the ST_LRCurve.

Definition

```
CREATE METHOD ST_Point
   (apoint ST_Point)
   RETURNS ST_PositionExp
   FOR ST_LRCurve
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_LRPosition(ST_Point)* takes the following input parameters:
   a) an *ST_Point* value *apoint*.
2) For the type-preserving method *ST_LRPosition(ST_Point)*:
   a) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) Return an *ST_PositionExp* value representing the linearly referenced location of a point on the *ST_LRCurve* closest to the given *ST_Point* value specified by *apoint* using the default Linear Referencing Method of the *ST_LRCurve*.
3) The returned position expression contains the *ST_LRCurve* value, its default LRM and the resultant measure value.
4) If the point is precisely on the *ST_LRCurve* value, or if the default LRM of the *ST_LRCurve* does not support offsets, then the returned position expression's distance expression will have no offset expression.
5) If the point is equidistant to more than one *ST_LRCurve* location, the one closest to the start of the *ST_LRCurve* is selected.

15.4.6 ST_LRCurveFromText Function

Purpose

Return an ST_LRCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRCurve value.

Definition

```
CREATE FUNCTION ST_LRCurveFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing feature value.

Description

1) The function *ST_LRCurveFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_LRCurveFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_LRCurve* value.

      If *awkt* is not producible in the BNF for <lr curve text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LRCurveFromText(awkt) AS ST_LRCurve)*.

15.4.7 ST_LRCurveFromGML Function

Purpose

Return an ST_LRCurve value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRCurve value.

Definition

```
CREATE FUNCTION ST_LRCurveFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_LRCurve
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_LRCurveFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_LRCurveFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain a curve type of LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LRCurveFromGML(agml) AS ST_LRCurve)*.

15.5 ST_LRDirectedEdge Type and Routines

15.5.1 ST_LRDirectedEdge Type

Purpose

The ST_LRDirectedEdge subtype of ST_LinearElement specifies any one-dimensional topology of type ST_Edge or ST_Link which can be linearly measured, that is, which supports the methods of ST_LinearElement. The ST_LRDirectedEdge type is instantiable.

Definition

```
CREATE TYPE ST_LRDirectedEdge
   UNDER ST_LinearElement
   AS (
      ST_PrivateTopologyType CHARACTER(1) DEFAULT NULL,
      ST_PrivateTopologyOrNetworkName
         CHARACTER VARYING(ST_MaxTopologyOrNetworkName) DEFAULT NULL,
      ST_PrivateEdgeOrLinkID INTEGER DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LRDirectedEdge
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
      RETURNS ST_LRDirectedEdge
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LRDirectedEdge
      (anleid INTEGER,
       anlrmid INTEGER,
       ameasure ST_LRMeasure,
       alinearelementtype CHARACTER VARYING(128),
       astartvaluearray ST_StartValue
          ARRAY[ST_MaxStartValueArrayElements],
       atopologytype CHARACTER(1),
       atopologyornetworkname
         CHARACTER VARYING(ST_MaxTopologyOrNetworkName),
       anedgeorlinkID INTEGER)
      RETURNS ST_LRDirectedEdge
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_TopologyType()
      RETURNS CHARACTER(1)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_TopologyType
   (atopologytype CHARACTER(1))
   RETURNS ST_LRDirectedEdge
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_TopoOrNetName()
   RETURNS CHARACTER VARYING(ST_MaxTopologyOrNetworkName)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_TopoOrNetName
   (atopologyornetworkname CHARACTER
    VARYING(ST_MaxTopologyOrNetworkName))
   RETURNS ST_LRDirectedEdge
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_EdgeOrLinkID()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_EdgeOrLinkID
   (anedgeorlinkID INTEGER)
   RETURNS ST_LRDirectedEdge
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxTopologyOrNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING topology or network name.
2) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
3) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.
4) The attribute *ST_PrivateTopologyType* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateTopologyType*.
5) The attribute *ST_PrivateTopologyOrNetworkName is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateTopologyOrNetworkName*.
6) The attribute *ST_PrivateEdgeOrLinkID is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateEdgeOrLinkID*.

Description

1) The *ST_LRDirectedEdge* type provides for public use:
   a) a method *ST_LRDirectedEdge(CHARACTER LARGE OBJECT)*,
   b) a method *ST_LRDirectedEdge(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER, CHARACTER VARYING, INTEGER)*,
   c) a method *ST_TopologyType()*,
   d) a method *ST_TopologyType(CHARACTER)*,
   e) a method *ST_TopoOrNetName()*,
   f) a method *ST_TopoOrNetName(CHARACTER VARYING)*,
   g) a method *ST_EdgeOrLinkID()*,
   h) a method *ST_EdgeOrLinkID(INTEGER)*,
   i) a function *ST_LREdgeFromText(CHARACTER LARGE OBJECT)*,
   j) a function *ST_LREdgeFromGML(CHARACTER LARGE OBJECT).*
2) The *ST_PrivateTopologyType* attribute contains a CHARACTER distinguishing the type of topology.
3) The *ST_PrivateTopologyOrNetworkName* attribute contains the CHARACTER VARYING topolopy or network name.
4) The *ST_PrivateEdgeOrLinkID* attribute contains the INTEGER edge ID or link ID.
5) The value of the *ST_PrivateTopologyType* attribute shall be 'E' (for edge) if the *ST_DirectedEdge* is part of a Topo-Geo topology.
6) The value of the *ST_PrivateTopologyType* attribute shall be 'L' (for link) if the *ST_DirectedEdge* is part of a Topo-Net network.
7) If the *ST_PrivateTopologyType* attribute equals 'E' (for edge), then the value of the *ST_PrivateTopologyOrNetworkName* attribute shall be the <topology-name> as specified in Clause 10, "Topology-Geometry".
8) If the *ST_PrivateTopologyType* attribute equals 'L' (for link), then the value of the *ST_PrivateTopologyOrNetworkName* attribute shall be the <network-name> as specified in Clause 11, "Topology-Network".
9) If the *ST_PrivateTopologyType* attribute equals 'E' (for edge), then the value of the *ST_PrivateEdgeOrLinkID* attribute shall be the EDGE ID as specified in Clause 10, "Topology-Geometry".
10) If the *ST_PrivateTopologyType* attribute equals 'L' (for link), then the value of the *ST_PrivateEdgeOrLinkID* attribute shall be the LINK ID as specified in Clause 11, "Topology-Network".

15.5.2 ST_LRDirectedEdge Methods

Purpose

Return an ST_LRDirectedEdge value constructed from either the well-known text representation; the GML representation; or the specified INTEGER leid, INTEGER default LRM lrmid, ST_LRMeasure default length, CHARACTER VARYING linear element type, ST_StartValue ARRAY, CHARACTER topology type, CHARACTER VARYING topology or network name and INTEGER edge or link ID values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LRDirectedEdge
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRDirectedEdge
   FOR ST_LRDirectedEdge
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_LRDirectedEdge
   (anleid INTEGER,
    anlrmid INTEGER,
    ameasure ST_LRMeasure,
    alinearelementtype CHARACTER VARYING(128),
    astartvaluearray ST_StartValue
      ARRAY[ST_MaxStartValueArrayElements],
    atopologytype CHARACTER(1),
    atopologyornetworkname CHARACTER VARYING(ST_MaxTopologyOrNetworkName),
    anedgeorlinkID INTEGER)
   RETURNS ST_LRDirectedEdge
   FOR ST_LRDirectedEdge
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyOrNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING topology or network name.
2) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
3) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.

Description

1) The method *ST_LRDirectedEdge(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) For the null-call type-preserving SQL-invoked constructor method *ST_LRDirectedEdge(CHARACTER LARGE OBJECT)*:

   Case:

   a) If *awktorgml* contains an edge type of LinearElement XML element in the GML representation, then return the result of the value expression: *ST_LREdgeFromGML(awktorgml)*.
   b) Otherwise, return the result of the value expression: *ST_LREdgeFromText(awktorgml)*.

3) The method *ST_LRDirectedEdge(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER, CHARACTER VARYING, INTEGER)* takes the following input parameters:

   a) an INTEGER value *anleid,*

   b) an INTEGER value *anlrmid,*

   c) an *ST_LRMeasure* value *ameasure,*

   d) a CHARACTER VARYING value *alinearelementtype,*

   e) an *ST_StartValue* ARRAY value *astartvaluearray,*

   f) a CHARACTER value *atopologytype,*

   g) a CHARACTER VARYING value *atopologyornetworkname,*

   h) an INTEGER value *anedgeorlinkID.*

4) The type-preserving SQL-invoked constructor method *ST_LRDirectedEdge(INTEGER, INTEGER, ST_LRMeasure, CHARACTER VARYING, ST_StartValue ARRAY, CHARACTER, CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *anleid* is the null value or if *anlrmid* is the null value or if *ameasure* is the null value or if *atopologytype* is the null value or if *atopologyornetworkname* is the null value or if *anedgeorlinkID* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRDirectedEdge* value with

      i) The *ST_PrivateLinearElementID* attribute set to *anleid.*

      ii) The *ST_PrivateDefaultLRM* attribute set to *anlrmid.*

      iii) The *ST_PrivateDefaultMeasure* attribute set to *ameasure.*

      iv) The *ST_PrivateLinearElementType* attribute set to *alinearelementtype.*

      v) The *ST_PrivateStartValues* attribute set to *astartvaluearray.*

      vi) The *ST_PrivateTopologyType* attribute set to *atopologytype*.

      vii) The *ST_PrivateTopologyOrNetworkName* attribute set to *atopologyornetworkname*.

      viii) The *ST_PrivateEdgeOrLinkID* attribute set to *anedgeorlinkID*.

15.5.3 ST_TopologyType Methods

Purpose

Observe and mutate the attribute ST_PrivateTopologyType of an ST_LRDirectedEdge value.

Definition

```
CREATE METHOD ST_TopologyType()
   RETURNS CHARACTER(1)
   FOR ST_LRDirectedEdge
   RETURN SELF.ST_PrivateTopologyType

CREATE METHOD ST_TopologyType
   (atopologytype CHARACTER(1))
   RETURNS ST_LRDirectedEdge
   FOR ST_LRDirectedEdge
   BEGIN
      IF atopologytype IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateTopologyType(atopologytype)
            END;
      END IF;
   END
```

Description

1) The method *ST_TopologyType()* has no input parameters.

2) The null-call method *ST_TopologyType()* returns the value of the *ST_PrivateTopologyType* attribute.

3) The method *ST_TopologyType(CHARACTER)* takes the following input parameters:

   a) a CHARACTER value *atopologytype*.

4) For the type-preserving method *ST_TopologyType(CHARACTER)*:

   Case:

   a) If *atopologytype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRDirectedEdge* value with the attribute *ST_PrivateTopologyType* set to *atopologytype*.

15.5.4 ST_TopoOrNetName Methods

Purpose

Observe and mutate the attribute ST_PrivateTopologyOrNetworkName of an ST_LRDirectedEdge value.

Definition

```
CREATE METHOD ST_TopoOrNetName()
   RETURNS CHARACTER VARYING(ST_MaxTopologyOrNetworkName)
   FOR ST_LRDirectedEdge
   RETURN SELF.ST_PrivateTopologyOrNetworkName

CREATE METHOD ST_TopoOrNetName
   (atopologyornetworkname CHARACTER VARYING(ST_MaxTopologyOrNetworkName))
   RETURNS ST_LRDirectedEdge
   FOR ST_LRDirectedEdge
   BEGIN
      IF atopologyornetworkname IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateTopologyOrNetworkName
                   (atopologyornetworkname)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxTopologyOrNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING topology or network name.

Description

1) The method *ST_TopoOrNetName()* has no input parameters.
2) The null-call method *ST_TopoOrNetName()* returns the value of the *ST_PrivateTopologyOrNetworkName* attribute.
3) The method *ST_TopoOrNetName(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopologyornetworkname*.
4) For the type-preserving method *ST_TopoOrNetName(CHARACTER VARYING)*:

   Case:

   a) If *atopologyornetworkname* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_LRDirectedEdge* value with the attribute *ST_PrivateTopologyOrNetworkName* set to *atopologyornetworkname*.

15.5.5 ST_EdgeOrLinkID Methods

Purpose

Observe and mutate the attribute ST_PrivateEdgeOrLinkID of an ST_LRDirectedEdge value.

Definition

```
CREATE METHOD ST_EdgeOrLinkID()
   RETURNS INTEGER
   FOR ST_LRDirectedEdge
   RETURN SELF.ST_PrivateEdgeOrLinkID

CREATE METHOD ST_EdgeOrLinkID
   (anedgeorlinkID INTEGER)
   RETURNS ST_LRDirectedEdge
   FOR ST_LRDirectedEdge
   BEGIN
      IF anedgeorlinkID IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateEdgeOrLinkID(anedgeorlinkID)
            END;
      END IF;
   END
```

Description

1) The method *ST_EdgeOrLinkID()* has no input parameters.

2) The null-call method *ST_EdgeOrLinkID()* returns the value of the *ST_PrivateEdgeOrLinkID* attribute.

3) The method *ST_EdgeOrLinkID(INTEGER)* takes the following input parameters:

   a) an INTEGER value *anedgeorlinkID*.

4) For the type-preserving method *ST_EdgeOrLinkID(INTEGER)*:

   Case:

   a) If *anedgeorlinkID* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRDirectedEdge* value with the attribute *ST_PrivateEdgeOrLinkID* set to *anedgeorlinkID*.

15.5.6 ST_LREdgeFromText Function

Purpose

Return an ST_LRDirectedEdge value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LRDirectedEdge value.

Definition

```
CREATE FUNCTION ST_LREdgeFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_LRDirectedEdge
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing feature value.

Description

1) The function *ST_LREdgeFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_LREdgeFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_LRDirectedEdge* value.

      If *awkt* is not producible in the BNF for <lr directed edge text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LREdgeFromText(awkt) AS ST_LRDirectedEdge)*.

15.5.7 ST_LREdgeFromGML Function

Purpose

Return an ST_LRDirectedEdge value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_LRDirectedEdge value.

Definition

```
CREATE FUNCTION ST_LREdgeFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_LRDirectedEdge
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_LREdgeFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_LREdgeFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain an edge type of LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_LREdgeFromGML(agml) AS ST_LRDirectedEdge)*.

15.6 ST_PositionExp Type and Routines

15.6.1 ST_PositionExp Type

Purpose

The ST_PositionExp type is used to specify a position as a linearly referenced location given by the linear element being measured, the method of measurement (LRM) and a measure value specified by a distance expression.

Definition

```
CREATE TYPE ST_PositionExp
   AS (
      ST_PrivateLinearElementID INTEGER DEFAULT NULL,
      ST_PrivateLinearElement ST_LinearElement DEFAULT NULL,
      ST_PrivateLRMID INTEGER DEFAULT NULL,
      ST_PrivateLRM ST_LRM DEFAULT NULL,
      ST_PrivateDistanceExpression ST_DistanceExp DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_PositionExp
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
      RETURNS ST_PositionExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_PositionExp
      (anleid INTEGER,
       anlrmid INTEGER,
       adistanceexpression ST_DistanceExp)
      RETURNS ST_PositionExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_PositionExp
      (anleid INTEGER,
       anlrm ST_LRM,
       adistanceexpression ST_DistanceExp)
      RETURNS ST_PositionExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_PositionExp
   (alinearelement ST_LinearElement,
    anlrmid INTEGER,
    adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_PositionExp
   (alinearelement ST_LinearElement,
    anlrm ST_LRM,
    adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LinearElementID()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LinearElementID
   (anleid INTEGER)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LinearElement()
   RETURNS ST_LinearElement
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LinearElement
   (alinearelement ST_LinearElement)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LRMID()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_LRMID
   (anlrmid INTEGER)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LRM()
   RETURNS ST_LRM
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LRM
   (anlrm ST_LRM)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_DistanceExp()
   RETURNS ST_DistanceExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DistanceExp
   (adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Equals
   (apositionexpression ST_PositionExp)
   RETURNS INTEGER
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
2) The attribute *ST_PrivateLinearElementID* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLinearElementID*.
3) The attribute *ST_PrivateLinearElement* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLinearElement*.
4) The attribute *ST_PrivateLRMID is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLRMID*.

5) The attribute *ST_PrivateLRM is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLRM*.

6) The attribute *ST_PrivateDistanceExpression is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateDistanceExpression*.

Description

1) The *ST_PositionExp* type provides for public use:
   a) a method *ST_PositionExp (CHARACTER LARGE OBJECT)*,
   b) a method *ST_PositionExp (INTEGER, INTEGER, ST_DistanceExp)*,
   c) a method *ST_PositionExp (INTEGER, ST_LRM, ST_DistanceExp)*,
   d) a method *ST_PositionExp (ST_LinearElement, INTEGER, ST_DistanceExp)*,
   e) a method *ST_PositionExp (ST_LinearElement, ST_LRM, ST_DistanceExp)*,
   f) a method *ST_LinearElementID()*,
   g) a method *ST_LinearElementID(INTEGER)*,
   h) a method *ST_LinearElement()*,
   i) a method *ST_LinearElement(ST_LinearElement)*,
   j) a method *ST_LRMID()*,
   k) a method *ST_LRMID(INTEGER)*,
   l) a method *ST_LRM()*,
   m) a method *ST_LRM(ST_LRM)*,
   n) a method *ST_DistanceExp()*,
   o) a method *ST_DistanceExp(ST_DistanceExp)*,
   p) a method *ST_Equals(ST_PositionExp)*,
   q) a function *ST_PosExpFromText(CHARACTER LARGE OBJECT)*,
   r) a function *ST_PosExpFromGML(CHARACTER LARGE OBJECT).*
2) The *ST_PrivateLinearElementID* attribute contains the INTEGER linear element ID (leid) value.
3) The *ST_PrivateLinearElement* attribute contains the *ST_LinearElement* linear element value.
4) The *ST_PrivateLRMID* attribute contains the INTEGER Linear Referencing Method ID (lrmid) value.
5) The *ST_PrivateLRM* attribute contains the *ST_LRM* Linear Referencing Method value.
6) The *ST_PrivateDistanceExpression* attribute contains the *ST_DistanceExp* distance expression value.
7) The type *ST_PositionExp* defines a single linearly referenced location.
8) The *ST_PrivateLinearElementID* attribute value shall not be NULL.
9) If the *ST_PrivateLinearElement* attribute value is not NULL, then the *ST_PrivateLinearElementID* attribute value shall be equal to the *ST_PrivateLinearElementID* of the *ST_LinearElement* that is the value of *ST_PrivateLinearElement.*
10) The *ST_PrivateLRMID* attribute value shall not be NULL.
11) If the *ST_PrivateLRM* attribute value is not NULL, then the *ST_PrivateLRMID* attribute value shall be equal to the *ST_PrivateLRMID* of the *ST_LRM* that is the value of *ST_PrivateLRM.*
12) For the *ST_Equals* method, two *ST_PositionExp* values are considered to be equal if they represent the same linearly referenced location*.* The two *ST_PositionExp* values may have different representations, such as different *ST_LRM*, *ST_LinearElement* and/or *ST_DistanceExp* values.

15.6.2 ST_PositionExp Methods

Purpose

Return an ST_PositionExp value constructed from either:

a) the well-known text representation;

b) the GML representation;

c) the specified INTEGER linear element leid, INTEGER Linear Referencing Method lrmid and ST_DistanceExp distance expression values;

d) the specified INTEGER linear element leid, ST_LRM Linear Referencing Method and ST_DistanceExp distance expression values;

e) the specified ST_LinearElement linear element, INTEGER Linear Referencing Method lrmid, and ST_DistanceExp distance expression values;

f) the specified ST_LinearElement linear element, ST_LRM Linear Referencing Method and ST_DistanceExp distance expression values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_PositionExp
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_PositionExp
   (anleid INTEGER,
    anlrmid INTEGER,
    adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_PositionExp
   (anleid INTEGER,
    anlrm ST_LRM,
    adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE CONSTRUCTOR METHOD ST_PositionExp
   (alinearelement ST_LinearElement,
    anlrmid INTEGER,
    adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_PositionExp
   (alinearelement ST_LinearElement,
    anlrm ST_LRM,
    adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.

Description

1) The method *ST_PositionExp (CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) For the null-call type-preserving SQL-invoked constructor method *ST_PositionExp (CHARACTER LARGE OBJECT)*:

   Case:
   a) If *awktorgml* contains a PositionExpression XML element in the GML representation, then return the result of the value expression: *ST_PosExpFromGML(awktorgml)*.
   b) Otherwise, return the result of the value expression: *ST_PosExpFromText(awktorgml)*.
3) The method *ST_PositionExp (INTEGER, INTEGER, ST_DistanceExp)* takes the following input parameters:
   a) an INTEGER value *anleid,*
   b) an INTEGER value *anlrmid,*
   c) an *ST_DistanceExp* value *adistanceexpression.*
4) The null-call type-preserving SQL-invoked constructor method *ST_PositionExp (INTEGER, INTEGER, ST_DistanceExp)* returns an *ST_PositionExp* value with:
   a) The *ST_PrivateLinearElementID* attribute set to *anleid.*
   b) The *ST_PrivateLRMID* attribute set to *anlrmid.*
   c) The *ST_PrivateDistanceExpression* attribute set to *adistanceexpression.*
5) The method *ST_PositionExp (INTEGER, ST_LRM, ST_DistanceExp)* takes the following input parameters:
   a) an INTEGER value *anleid,*
   b) an *ST_LRM* value *anlrm,*
   c) an *ST_DistanceExp* value *adistanceexpression.*

6) The null-call type-preserving SQL-invoked constructor method *ST_PositionExp (INTEGER, ST_LRM, ST_DistanceExp)* returns an *ST_PositionExp* value with:
   a) The *ST_PrivateLinearElementID* attribute set to *anleid.*
   b) The *ST_PrivateLRM* attribute set to *anlrm.*
   c) The *ST_PrivateLRMID* attribute set to *anlrm.ST_LRMID().*
   d) The *ST_PrivateDistanceExpression* attribute set to *adistanceexpression.*
7) The method *ST_PositionExp (ST_LinearElement, INTEGER, ST_DistanceExp)* takes the following input parameters:
   a) an *ST_LinearElement* value *alinearelement,*
   b) an INTEGER value *anlrmid,*
   c) an *ST_DistanceExp* value *adistanceexpression.*
8) The null-call type-preserving SQL-invoked constructor method *ST_PositionExp (ST_LinearElement, INTEGER, ST_DistanceExp)* returns an *ST_PositionExp* value with:
   a) The *ST_PrivateLinearElement* attribute set to *alinearelement.*
   b) The *ST_PrivateLinearElementID* attribute set to *alinearelement.ST_LinearElementID().*
   c) The *ST_PrivateLRMID* attribute set to *anlrmid.*
   d) The *ST_PrivateDistanceExpression* attribute set to *adistanceexpression.*
9) The method *ST_PositionExp (ST_LinearElement, ST_LRM, ST_DistanceExp)* takes the following input parameters:
   a) an *ST_LinearElement* value *alinearelement,*
   b) an *ST_LRM* value *anlrm,*
   c) an *ST_DistanceExp* value *adistanceexpression.*
10) The null-call type-preserving SQL-invoked constructor method *ST_PositionExp (ST_LinearElement, INTEGER, ST_DistanceExp)* returns an *ST_PositionExp* value with:
   a) The *ST_PrivateLinearElement* attribute set to *alinearelement.*
   b) The *ST_PrivateLinearElementID* attribute set to *alinearelement.ST_LinearElementID().*
   c) The *ST_PrivateLRM* attribute set to *anlrm.*
   d) The *ST_PrivateLRMID* attribute set to *anlrm.ST_LRMID().*
   e) The *ST_PrivateDistanceExpression* attribute set to *adistanceexpression.*

15.6.3 ST_LinearElementID Methods

Purpose

Observe and mutate the attribute ST_PrivateLinearElementID of an ST_PositionExp value.

Definition

```
CREATE METHOD ST_LinearElementID()
   RETURNS INTEGER
   FOR ST_PositionExp
   RETURN SELF.ST_PrivateLinearElement

CREATE METHOD ST_LinearElementID
   (anleid INTEGER)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      IF anleid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLinearElementID(anleid)
            END;
      END IF;
   END
```

Description

1) The method *ST_LinearElementID()* has no input parameters.

2) The null-call method *ST_LinearElementID()* returns the value of the *ST_PrivateLinearElementID* attribute.

3) The method *ST_LinearElementID(INTEGER)* takes the following input parameters:

   a) an INTEGER value *anleid*.

4) For the type-preserving method *ST_LinearElementID(INTEGER)*:

   Case:

   a) If *anleid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateLinearElementID* set to *anleid*.

15.6.4 ST_LinearElement Methods

Purpose

Observe and mutate the attribute ST_PrivateLinearElement of an ST_PositionExp value.

Definition

```
CREATE METHOD ST_LinearElement()
   RETURNS ST_LinearElement
   FOR ST_PositionExp
   RETURN SELF.ST_PrivateLinearElement

CREATE METHOD ST_LinearElement
   (alinearelement ST_LinearElement)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      IF alinearelement IS NULL THEN
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLinearElement(alinearelement)
            END;
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLinearElement(alinearelement).
                     ST_PrivateLinearElementID
                      (alinearelement.ST_LinearElementID())
            END;
      END IF;
   END
```

Description

1) The method *ST_LinearElement()* has no input parameters.

2) The null-call method *ST_LinearElement()* returns the value of the *ST_PrivateLinearElement* attribute.

3) The method *ST_LinearElement(ST_LinearElement)* takes the following input parameters:

   a) an *ST_LinearElement* value *alinearelement*.

4) For the type-preserving method *ST_LinearElement(ST_LinearElement)*:

   Case:

   a) If *alinearelement* is the null value, then:

      i) If SELF is the null value, then return the null value.

      ii) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateLinearElement* set to *alinearelement*.

   b) Otherwise:

      i) If SELF is the null value, then return the null value.

      ii) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateLinearElement* set to *alinearelement* and the *ST_PrivateLinearElementID* set to *alinearelement.ST_LinearElementID()*.

15.6.5 ST_LRMID Methods

Purpose

Observe and mutate the attribute ST_PrivateLRMID of an ST_PositionExp value.

Definition

```
CREATE METHOD ST_LRMID()
   RETURNS INTEGER
   FOR ST_PositionExp
   RETURN SELF.ST_PrivateLRMID

CREATE METHOD ST_LRMID
   (anlrmid INTEGER)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      IF anlrmid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRMID(anlrmid)
            END;
      END IF;
   END
```

Description

1) The method *ST_LRMID()* has no input parameters.

2) The null-call method *ST_LRMID()* returns the value of the *ST_PrivateLRMID* attribute.

3) The method *ST_LRMID(INTEGER)* takes the following input parameters:

   a) an INTEGER value *anlrmid.*

4) For the type-preserving method *ST_LRMID(INTEGER)*:

   Case:

   a) If *anlrmid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateLRMID* set to *anlrmid.*

15.6.6 ST_LRM Methods

Purpose

Observe and mutate the attribute ST_PrivateLRM of an ST_PositionExp value.

Definition

```
CREATE METHOD ST_LRM()
   RETURNS ST_LRM
   FOR ST_PositionExp
   RETURN SELF.ST_PrivateLRM

CREATE METHOD ST_LRM
   (anlrm ST_LRM)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      IF anlrm IS NULL THEN
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRM(anlrm)
            END;
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRM(anlrm).
                     ST_PrivateLRMID(anlrm.ST_LRMID())
            END;
      END IF;
   END
```

Description

1) The method *ST_LRM()* has no input parameters.

2) The null-call method *ST_LRM()* returns the value of the *ST_PrivateLRM* attribute.

3) The method *ST_LRM(ST_LRM)* takes the following input parameters:

   a) an *ST_LRM* value *anlrm.*

4) For the type-preserving method *ST_LRM(ST_LRM)*:

   Case:

   a) If *anlrm* is the null value, then:

      i) If SELF is the null value, then return the null value.

      ii) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateLRM* set to *anlrm*.

   b) Otherwise:

      i) If SELF is the null value, then return the null value.

      ii) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateLRM* set to *anlrm* and the *ST_PrivateLRMID* set to *anlrm.ST_LRMID()*.

15.6.7 ST_DistanceExp Methods

Purpose

Observe and mutate the attribute ST_PrivateDistanceExpression of an ST_PositionExp value.

Definition

```
CREATE METHOD ST_DistanceExp()
   RETURNS ST_DistanceExp
   FOR ST_PositionExp
   RETURN SELF.ST_PrivateDistanceExpression

CREATE METHOD ST_DistanceExp
   (adistanceexpression ST_DistanceExp)
   RETURNS ST_PositionExp
   FOR ST_PositionExp
   BEGIN
      IF adistanceexpression IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateDistanceExpresssion(adistanceexpression)
            END;
      END IF;
   END
```

Description

1) The method *ST_DistanceExp()* has no input parameters.

2) The null-call method *ST_DistanceExp()* returns the value of the *ST_PrivateDistanceExpression* attribute.

3) The method *ST_DistanceExp(ST_DistanceExp)* takes the following input parameters:

   a) an *ST_DistanceExp* value *adistanceexpression*.

4) For the type-preserving method *ST_DistanceExp(ST_DistanceExp)*:

   Case:

   a) If *adistanceexpression* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_PositionExp* value with the attribute *ST_PrivateDistanceExpression* set to *adistanceexpression*.

15.6.8 ST_Equals Method

Purpose

Test if an ST_PositionExp specifies the same linearly referenced location as another ST_PositionExp value.

Definition

```
CREATE METHOD ST_Equals
   (apositionexpression ST_PositionExp)
   RETURNS INTEGER
   FOR ST_PositionExp
   BEGIN
      IF apositionexpression IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  --
                  -- See Description
                  --
            END;
      END IF;
   END
```

Description

1) The method *ST_Equals(ST_*PositionExp*)* takes the following input parameters:

   a) an *ST_PositionExp* value *apositionexpression*.

2) For the method *ST_Equals(ST_*PositionExp*)*:

   Case:

   a) If *apositionexpression* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise,

      i) If it is not possible to determine whether the two position expressions represent the same linearly referenced location, then an exception condition is raised: *SQL/MM Spatial exception – indeterminate equality*.

      ii) If SELF and *apositionexpression* specify the same linearly referenced location, then return 1 (one).

      iii) Otherwise, return 0 (zero).

15.6.9 ST_PosExpFromText Function

Purpose

Return an ST_PositionExp value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_PositionExp value.

Definition

```
CREATE FUNCTION ST_PosExpFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_PositionExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing type value.

Description

1) The function *ST_PosExpFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_PosExpFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_PositionExp* value.

      If *awkt* is not producible in the BNF for <position expression text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_PosExpFromText(awkt) AS ST_PositionExp)*.

15.6.10 ST_PosExpFromGML Function

Purpose

Return an ST_PositionExp value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML PositionExpression representation of an ST_PositionExp value.

Definition

```
CREATE FUNCTION ST_PosExpFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_PositionExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_PosExpFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_PosExpFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain a PositionExpression XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_PosExpFromGML(agml) AS ST_PositionExp)*.

15.7 ST_LRMeasure Type and Routines

15.7.1 ST_LRMeasure Type

Purpose

The ST_LRMeasure type specifies a measured value with optional units of measure.

Definition

```
CREATE TYPE ST_LRMeasure
   AS (
      ST_PrivateMeasure DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateUnits CHARACTER VARYING(ST_MaxUnitNameLength) DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LRMeasure
      (ameasure DOUBLE PRECISION)
      RETURNS ST_LRMeasure
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LRMeasure
      (ameasure DOUBLE PRECISION,
       aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
      RETURNS ST_LRMeasure
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_Measure()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Measure
      (ameasure DOUBLE PRECISION)
      RETURNS ST_LRMeasure
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_UnitOfMeasure()
      RETURNS CHARACTER VARYING(ST_MaxUnitNameLength)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_UnitOfMeasure
   (aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRMeasure
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.
2) The attribute *ST_PrivateMeasure is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateMeasure*.
3) The attribute *ST_PrivateUnits is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateUnits*.

Description

1) The *ST_LRMeasure* type provides for public use:
   a) a method *ST_LRMeasure(DOUBLE PRECISION)*,
   b) a method *ST_LRMeasure(DOUBLE PRECISION, CHARACTER VARYING)*,
   c) a method *ST_Measure()*,
   d) a method *ST_Measure(DOUBLE PRECISION)*,
   e) a method *ST_UnitOfMeasure()*,
   f) a method *ST_UnitOfMeasure(CHARACTER VARYING)*.
2) The *ST_PrivateMeasure* attribute contains the DOUBLE PRECISION measured value.
3) The *ST_PrivateUnits* attribute contains the optional CHARACTER VARYING units of measure value.
4) The allowable values specified by the *ST_PrivateUnits* attribute shall be a supported <unit name>. The value is a supported <unit name> if and only if the value is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.

15.7.2 ST_LRMeasure Methods

Purpose

Return an ST_LRMeasure value constructed from either the specified DOUBLE PRECISION measure value or the specified DOUBLE PRECISION measure and CHARACTER VARYING unit of measure values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LRMeasure
   (ameasure DOUBLE PRECISION)
   RETURNS ST_LRMeasure
   FOR ST_LRMeasure
   RETURN NEW ST_LRMeasure(ameasure, NULL)

CREATE CONSTRUCTOR METHOD ST_LRMeasure
   (ameasure DOUBLE PRECISION,
    aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRMeasure
   FOR ST_LRMeasure
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_LRMeasure(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *ameasure.*
2) The null-call type-preserving SQL-invoked constructor method *ST_LRMeasure(DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_LRMeasure(ameasure, NULL).*
3) The method *ST_LRMeasure(DOUBLE PRECISION, CHARACTER VARYING)* takes the following input parameters:
   a) a DOUBLE PRECISION value *ameasure,*
   b) a CHARACTER VARYING value *aunitofmeasure*.
4) For the type-preserving SQL-invoked constructor method *ST_LRMeasure(DOUBLE PRECISION, CHARACTER VARYING)*:

   Case:

   a) If *ameasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, if *aunitofmeasure* is NOT NULL, then:
      i) The values for *aunitofmeasure* shall be a supported <unit name>.
      ii) The value for *aunitofmeasure* is a supported <unit name> if and only if the value of *aunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.
      iii) If the unit specified by *aunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.

d) Return an *ST_LRMeasure* value with:

   i) The *ST_PrivateMeasure* attribute set to *ameasure*.

   ii) The *ST_PrivateUnits* attribute set to *aunitofmeasure.*

15.7.3 ST_Measure Methods

Purpose

Observe and mutate the attribute ST_PrivateMeasure of an ST_LRMeasure value.

Definition

```
CREATE METHOD ST_Measure()
   RETURNS DOUBLE PRECISION
   FOR ST_LRMeasure
   RETURN SELF.ST_PrivateMeasure

CREATE METHOD ST_Measure
   (ameasure DOUBLE PRECISION)
   RETURNS ST_LRMeasure
   FOR ST_LRMeasure
   BEGIN
      IF ameasure IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateMeasure(ameasure)
            END;
      END IF;
   END
```

Description

1) The method *ST_Measure()* has no input parameters.

2) The null-call method *ST_Measure()* returns the value of the *ST_PrivateMeasure* attribute.

3) The method *ST_Measure(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *ameasure*.

4) For the type-preserving method *ST_Measure(DOUBLE PRECISION)*:

   Case:

   a) If *ameasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LRMeasure* value with the attribute *ST_PrivateMeasure* set to *ameasure*.

15.7.4 ST_UnitOfMeasure Methods

Purpose

Observe and mutate the attribute ST_PrivateUnits of an ST_LRMeasure value.

Definition

```
CREATE METHOD ST_UnitOfMeasure()
   RETURNS CHARACTER VARYING(ST_MaxUnitNameLength)
   FOR ST_LRMeasure
   RETURN SELF.ST_PrivateUnits

CREATE METHOD ST_UnitOfMeasure
   (aunitofmeasure CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_LRMeasure
   FOR ST_LRMeasure
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            --
            -- See Description
            --
      END
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

Description

1) The method *ST_UnitOfMeasure()* has no input parameters.
2) The null-call method *ST_UnitOfMeasure()* returns the value of the *ST_PrivateUnits* attribute.
3) The method *ST_UnitOfMeasure(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *aunitofmeasure*.
4) For the type-preserving method *ST_UnitOfMeasure(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise:
      i) The values for *aunitofmeasure* shall be a supported <unit name>.
      ii) The value for *aunitofmeasure* is a supported <unit name> if and only if the value of *aunitofmeasure* is equal to the value of the UNIT_NAME column of one of the rows where the value of the UNIT_TYPE column is equal to 'LINEAR' in the ST_INFORMTN_SCHEMA ST_UNITS_OF_MEASURE view.
      iii) If the unit specified by *aunitofmeasure* is not supported by the implementation, then an exception condition is raised: *SQL/MM Spatial exception – unsupported unit specified*.
      iv) Return an *ST_LRMeasure* value with the attribute *ST_PrivateUnits* set to *aunitofmeasure*.

15.8 ST_StartValue Type and Routines

15.8.1 ST_StartValue Type

Purpose

The ST_StartValue type specifies an LRM lrmid and its start measure value for a particular ST_LinearElement.

Definition

```
CREATE TYPE ST_StartValue
   AS (
      ST_PrivateLRM INTEGER DEFAULT NULL,
      ST_PrivateMeasure ST_LRMeasure DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_StartValue
      (anlrmid INTEGER,
       ameasure ST_LRMeasure)
      RETURNS ST_StartValue
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_LRM()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_LRM
      (anlrmid INTEGER)
      RETURNS ST_StartValue
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_LRMeasure()
      RETURNS ST_LRMeasure
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Measure
      (ameasure ST_LRMeasure)
      RETURNS ST_StartValue
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateLRM is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLRM*.
2) The attribute *ST_PrivateMeasure is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateMeasure*.

Description

1) The *ST_StartValue* type provides for public use:
   a) a method *ST_StartValue (INTEGER, ST_LRMeasure)*,
   b) a method *ST_LRM()*,
   c) a method *ST_LRM(INTEGER)*,
   d) a method *ST_Measure()*,
   e) a method *ST_Measure(ST_LRMeasure)*.
2) The *ST_PrivateLRM* attribute contains the INTEGER lrmid value of the Linear Referencing Method which defines how the measurement is made.
3) The *ST_PrivateMeasure* attribute contains the *ST_LRMeasure* measure value at the start of a *ST_LinearElement*.
4) If the *ST_PrivateUnits* attribute of the *ST_LRMeasure* value specified by the *ST_PrivateMeasure* attribute is NULL, then the *ST_PrivateUnits* attribute value of the *ST_LRM* value specified by the *ST_PrivateLRM* attribute of the *ST_StartValue* value shall apply.

15.8.2 ST_StartValue Method

Purpose

Return an ST_StartValue value constructed from the specified INTEGER Linear Referencing Method lrmid and ST_LRMeasure measure values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_StartValue
   (anlrmid INTEGER,
    ameasure ST_LRMeasure)
   RETURNS ST_StartValue
   FOR ST_StartValue
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_StartValue(INTEGER, ST_LRMeasure)* takes the following input parameters:
   a) an INTEGER value *anlrmid,*
   b) an *ST_LRMeasure* value *ameasure*.
2) The null-call type-preserving SQL-invoked constructor method *ST_StartValue(INTEGER, ST_LRMeasure)* returns an *ST_StartValue* value with:
   a) The *ST_PrivateLRM* attribute set to *anlrmid.*
   b) The *ST_PrivateMeasure* attribute set to *ameasure.*

15.8.3 ST_LRM Methods

Purpose

Observe and mutate the attribute ST_PrivateLRM of an ST_StartValue value.

Definition

```
CREATE METHOD ST_LRM()
   RETURNS INTEGER
   FOR ST_StartValue
   RETURN SELF.ST_PrivateLRM

CREATE METHOD ST_LRM
   (anlrmid INTEGER)
   RETURNS ST_StartValue
   FOR ST_StartValue
   BEGIN
      IF anlrmid IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLRM(anlrmid)
            END;
      END IF;
   END
```

Description

1) The method *ST_LRM()* has no input parameters.
2) The null-call method *ST_LRM()* returns the value of the *ST_PrivateLRM* attribute.
3) The method *ST_LRM(INTEGER)* takes the following input parameters:
   a) an INTEGER value *anlrmid*.
4) For the type-preserving method *ST_LRM(INTEGER)*:

   Case:

   a) If *anlrmid* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_StartValue* value with the attribute *ST_PrivateLRM* set to *anlrmid*.

15.8.4 ST_Measure Methods

Purpose

Observe and mutate the attribute ST_PrivateMeasure of an ST_StartValue value.

Definition

```
CREATE METHOD ST_Measure()
   RETURNS ST_LRMeasure
   FOR ST_StartValue
   RETURN SELF.ST_PrivateMeasure

CREATE METHOD ST_Measure
   (ameasure ST_LRMeasure)
   RETURNS ST_StartValue
   FOR ST_StartValue
   BEGIN
      IF ameasure IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateMeasure(ameasure)
            END;
      END IF;
   END
```

Description

1) The method *ST_Measure()* has no input parameters.

2) The null-call method *ST_Measure()* returns the value of the *ST_PrivateMeasure* attribute.

3) The method *ST_Measure(ST_LRMeasure)* takes the following input parameters:

   a) an *ST_LRMeasure* value *ameasure*.

4) For the type-preserving method *ST_Measure(ST_LRMeasure)*:

   Case:

   a) If *ameasure* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_StartValue* value with the attribute *ST_PrivateMeasure* set to *ameasure*.

15.9 ST_DistanceExp Type and Routines

15.9.1 ST_DistanceExp Type

Purpose

The ST_DistanceExp type specifies the linear referenced measure value. The ST_DistanceExp type is instantiable.

Definition

```
CREATE TYPE ST_DistanceExp
   AS (
      ST_PrivateDistanceAlong ST_LRMeasure DEFAULT NULL,
      ST_PrivateFromReferentFeatureID
         CHARACTER VARYING(MaxFeatureIDLength) DEFAULT NULL,
      ST_PrivateFromReferentName
         CHARACTER VARYING(ST_MaxReferentNameLength) DEFAULT NULL,
      ST_PrivateTowardsReferentFeatureID
         CHARACTER VARYING(MaxFeatureIDLength) DEFAULT NULL,
      ST_PrivateTowardsReferentName
         CHARACTER VARYING(ST_MaxReferentNameLength) DEFAULT NULL,
      ST_PrivateLateralOffsetExpression ST_LatOffsetExp DEFAULT NULL,
      ST_PrivateVerticalOffsetExpression ST_VerOffsetExp DEFAULT NULL,
      ST_PrivateVectorOffsetExpression ST_VectorOffsetExp DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_DistanceExp
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
      RETURNS ST_DistanceExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_DistanceExp
      (adistancealong ST_LRMeasure)
      RETURNS ST_DistanceExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_DistanceExp
      (adistancealong ST_LRMeasure,
       alateraloffsetexpression ST_LatOffsetExp)
      RETURNS ST_DistanceExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    alateraloffsetexpression ST_LatOffsetExp,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_DistanceAlong()
   RETURNS ST_LRMeasure
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DistanceAlong
   (adistancealong ST_LRMeasure)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_FromRefFeaID()
   RETURNS CHARACTER VARYING(MaxFeatureIDLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_FromRefFeaID
   (afromreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength))
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_FromRefName()
   RETURNS CHARACTER VARYING(ST_MaxReferentNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_FromRefName
   (afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_TowardsRefFeaID()
   RETURNS CHARACTER VARYING(MaxFeatureIDLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_TowardsRefFeaID
   (atowardsreferentfeatureID CHARACTER VARYING(MaxFeatureIDLength))
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_TowardsRefName()
   RETURNS CHARACTER VARYING(ST_MaxReferentNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_TowardsRefName
   (atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_LatOffsetExp()
   RETURNS ST_LatOffsetExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_LatOffsetExp
   (alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_VerOffsetExp()
   RETURNS ST_VerOffsetExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_VerOffsetExp
   (averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_VectorOffsetExp()
   RETURNS ST_VectorOffsetExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_VectorOffsetExp
   (avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
2) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.
3) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.
4) The attribute *ST_PrivateDistanceAlong is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateDistanceAlong*.
5) The attribute *ST_PrivateFromReferentFeatureID is* not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateFromReferentFeatureID*.
6) The attribute *ST_PrivateFromReferentName* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateFromReferentName*.
7) The attribute *ST_PrivateTowardsReferentFeatureID* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateTowardsReferentFeatureID*.
8) The attribute *ST_PrivateTowardsReferentName* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateTowardsReferentName*.
9) The attribute *ST_PrivateLateralOffsetExpression* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLateralOffsetExpression*.
10) The attribute *ST_PrivateVerticalOffsetExpression* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateVerticalOffsetExpression*.
11) The attribute *ST_PrivateVectorOffsetExpression* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateVectorOffsetExpression*.

Description

1) The *ST_DistanceExp* type provides for public use:
   a) a method *ST_DistanceExp(CHARACTER LARGE OBJECT)*,
   b) a method *ST_DistanceExp(ST_LRMeasure)*,
   c) a method *ST_DistanceExp(ST_LRMeasure, ST_LatOffsetExp)*,
   d) a method *ST_DistanceExp(ST_LRMeasure, ST_VerOffsetExp)*,

e) a method *ST_DistanceExp(ST_LRMeasure, ST_LatOffsetExp, ST_VerOffsetExp)*,

f) a method *ST_DistanceExp(ST_LRMeasure, ST_VectorOffsetExp)*,

g) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING)*,

h) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp)*,

i) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_VerOffsetExp)*,

j) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp, ST_VerOffsetExp)*,

k) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_VectorOffsetExp)*,

l) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING)*,

m) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp)*,

n) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_VerOffsetExp)*,

o) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp, ST_VerOffsetExp)*,

p) a method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_VectorOffsetExp)*,

q) a method *ST_DistanceAlong()*,

r) a method *ST_DistanceAlong(ST_LRMeasure)*,

s) a method *ST_FromRefFeaID()*,

t) a method *ST_FromRefFeaID(CHARACTER VARYING)*,

u) a method *ST_FromRefName()*,

v) a method *ST_FromRefName(CHARACTER VARYING)*,

w) a method *ST_TowardsRefFeaID()*,

x) a method *ST_TowardsRefFeaID(CHARACTER VARYING)*,

y) a method *ST_TowardsRefName()*,

z) a method *ST_TowardsRefName(CHARACTER VARYING)*,

aa) a method *ST_LatOffsetExp()*,

ab) a method *ST_LatOffsetExp(ST_LatOffsetExp)*,

ac) a method *ST_VerOffsetExp()*,

ad) a method *ST_VerOffsetExp(ST_VerOffsetExp)*,

ae) a method *ST_VectorOffsetExp()*,

af) a method *ST_VectorOffsetExp(ST_VectorOffsetExp)*,

ag) a function *ST_DisExpFromText(CHARACTER LARGE OBJECT)*,

ah) a function *ST_DisExpFromGML(CHARACTER LARGE OBJECT).*

2) The *ST_PrivateDistanceAlong* attribute contains the *ST_LRMeasure* distance along measure of the distance expression.

3) The *ST_PrivateFromReferentFeatureID* attribute optionally contains the CHARACTER VARYING feature ID of the feature owning the "from" referent.

4) The *ST_PrivateFromReferentName* attribute optionally contains the CHARACTER VARYING "from" referent name of the distance expression.

5) The *ST_PrivateTowardsReferentFeatureID* attribute optionally contains the CHARACTER VARYING feature ID of the feature owning the "towards" referent.

6) The *ST_PrivateTowardsReferentName* attribute optionally contains the CHARACTER VARYING "towards" referent name of the distance expression.

7) The *ST_PrivateLateralOffsetExpression* attribute optionally contains the *ST_LatOffsetExp* lateral offset expression of the distance expression.

8) The *ST_PrivateVerticalOffsetExpression* attribute optionally contains the *ST_VerOffsetExp* vertical offset expression of the distance expression.

9) The *ST_PrivateVectorOffsetExpression* attribute optionally contains the *ST_VectorOffsetExp* vector offset expression of the distance expression.

10) The *ST_DistanceExp* specifies the measure along (and sometimes offset from) an *ST_LinearElement* associated with the *ST_Distance Exp*.

11) The semantics of the *ST_DistanceExp* value are dictated by the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp*.

12) The *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* is either explicitly specified by the *ST_PrivateLRMID* attribute of the *ST_PositionExp* owning the *ST_DistanceExp* or implicit in how and where this *ST_DistanceExp* is used.

13) The *ST_LinearElement* associated with the *ST_DistanceExp* is either explicitly specified by the *ST_PrivateLinearElementID* attribute of the *ST_PositionExp* owning the *ST_DistanceExp* or implicit in how and where this *ST_DistanceExp* is used.

14) An *ST_DistanceExp* shall contain a mandatory distance along *ST_LRMeasure* value.

15) If the *ST_PrivateUnits* attribute of the *ST_LRMeasure* value specified by the *ST_PrivateDistanceAlong* attribute is NULL, then the *ST_PrivateUnits* attribute value of the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* shall apply.

16) If the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* has an *ST_PrivateLRMType* attribute value equal to "absolute", then the distance along value of the *ST_DistanceExp* is measured from the start of the associated *ST_LinearElement* in the direction of the *ST_Linear Element*.

    a) If there exists an *ST_StartValue* element in the *ST_PrivateStartValues* attribute *ST_StartValues* ARRAY of the associated *ST_LinearElement* having an *ST_PrivateLRM* value equal to the *ST_PrivateLRMID* attribute value of the associated *ST_LRM*, then

        i) Let *SV* be that *ST_StartValue* value.

        ii) Let *M* be the *ST_PrivateMeasure* attribute *ST_LRMeasure* value of *SV*.

        iii) In determining the linearly referenced location along the *ST_LinearElement*, the distance along *ST_LRMeasure* value of the *ST_DistanceExp* shall be modified by *M*.

17) If the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* has an *ST_PrivateLRMType* attribute value equal to "interpolative", then the distance along value of the *ST_DistanceExp* is pro-rated, based upon the *ST_PrivateDefaultMeasure* attribute value of the associated *ST_LinearElement*.

18) If the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* has an *ST_PrivateLRMType* attribute value equal to "local interpolative", then the distance along value of the *ST_DistanceExp* is interpolated locally along a segment of a curve type linear element bounded by two control points having m coordinate values bracketing the distance along measure value.

19) If and only if the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* has an *ST_PrivateLRMType* attribute value equal to "relative", then the *ST_DistanceExp* may have a "from" or a "from" and a "towards" referent.

20) If the *ST_DistanceExp* has a "from" referent and no "towards" referent, the distance along value of the *ST_DistanceExp* is measured from the "from" referent location on the *ST_LinearElement* in the direction of the *ST_Linear Element*.

21) If the *ST_DistanceExp* has a "from" referent and a "towards" referent, the distance along value of the *ST_DistanceExp* is measured from the "from" referent location on the *ST_LinearElement* in a direction along the *ST_Linear Element* towards the "towards" referent.

22) If the *ST_DistanceExp* has a "towards" referent, then it shall also have a "from" referent.

23) If the *ST_PrivateFromReferentFeatureID* attribute value is NULL, then a feature ID value equal to the *ST_PrivateFeatureID* attribute value of the *ST_LRFeature* associated with the *ST_DistanceExp* SELF shall apply.

24) If the *ST_PrivateTowardsReferentFeatureID* attribute value is NULL, then a feature ID value equal to the *ST_PrivateFeatureID* attribute value of the *ST_LRFeature* associated with the *ST_DistanceExp* SELF shall apply.

25) If the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* allows offsets, then the *ST_DistanceExp* shall have a lateral offset expression, a vertical offset expression, a lateral and a vertical offset expression, a vector offset expression, or no offset expression.

26) If the *ST_DistanceExp* contains a lateral offset expression, then the direction of the lateral measure is specified by the *ST_PrivatePositiveLateralOffsetDirection* of the *ST_LRM* associated with the *ST_DistanceExp*. If this is NULL, a default value of ‘right’ shall apply.

27) If the *ST_DistanceExp* contains a vertical offset expression, then the direction of the vertical measure is specified by the *ST_PrivatePositiveVerticalOffsetDirection* of the *ST_LRM* associated with the *ST_DistanceExp*. If this is NULL, a default value of ‘up’ shall apply.

28) If the *ST_DistanceExp* has a lateral offset expression, the linearly referenced location is determined by first measuring the distance along along the *ST_LinearElement* and then applying the optional lateral offset expression value.

29) If the *ST_DistanceExp* has a vertical offset expression, the linearly referenced location is determined by first measuring the distance along along the *ST_LinearElement* and then applying the optional vertical offset expression value.

30) If the *ST_DistanceExp* has a vector offset expression, the linearly referenced location is determined by first measuring the distance along along the *ST_LinearElement* and then applying the optional vector offset expression value.

31) If the *ST_DistanceExp* has no offset expressions, the linearly referenced location is on the *ST_LinearElement* at a location determined by measuring the distance along along the *ST_LinearElement*.

32) The feature specified by a not NULL value for the attributes *ST_PrivateFromReferentFeatureID* or *ST_PrivateTowardsReferentFeatureID* do not have to be the same *ST_LinearElement* specified by the *ST_LinearElement* associated with the *ST_DistanceExp* SELF.

15.9.2 ST_DistanceExp Methods

Purpose

Return an ST_DistanceExp value constructed from either:

a) the well-known text representation;

b) the GML representation;

c) the specified ST_LRMeasure distance along value;

d) the specified ST_LRMeasure distance along and ST_LatOffsetExp lateral offset expression values;

e) e specified ST_LRMeasure distance along and ST_VerOffsetExp vertical offset expression values;

f) e specified ST_LRMeasure distance along, ST_LatOffsetExp lateral offset expression and ST_VerOffsetExp vertical offset expression values;

g) the specified ST_LRMeasure distance along and ST_VectorOffsetExp vector offset expression values;

h) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID and CHARACTER VARYING "from" referent name values;

i) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name and ST_LatOffsetExp lateral offset expression values;

j) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name and ST_VerOffsetExp vertical offset expression values;

k) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name, ST_LatOffsetExp lateral offset expression and ST_VerOffsetExp vertical offset expression values;

l) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name and ST_VectorOffsetExp vector offset expression values;

m) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name, CHARACTER VARYING "towards" referent feature ID and CHARACTER VARYING "towards" referent name values;

n) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name, CHARACTER VARYING "towards" referent feature ID, CHARACTER VARYING "towards" referent name and ST_LatOffsetExp lateral offset expression values;

o) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name, CHARACTER VARYING "towards" referent feature ID, CHARACTER VARYING "towards" referent name and ST_VerOffsetExp vertical offset expression values;

p) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name, CHARACTER VARYING "towards" referent feature ID, CHARACTER VARYING "towards" referent name, ST_LatOffsetExp lateral offset expression and ST_VerOffsetExp vertical offset expression values;

q) the specified ST_LRMeasure distance along, CHARACTER VARYING "from" referent feature ID, CHARACTER VARYING "from" referent name, CHARACTER VARYING "towards" referent feature ID, CHARACTER VARYING "towards" referent name and ST_VectorOffsetExp vector offset expression values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, NULL, NULL, NULL, NULL, NULL,
    NULL)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, NULL, NULL, NULL, NULL,
    alateraloffsetexpression, NULL)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, NULL, NULL, NULL, NULL,
    NULL, averticaloffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    alateraloffsetexpression ST_LatOffsetExp,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, NULL, NULL, NULL, NULL,
    alateraloffsetexpression, averticaloffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, NULL, NULL, NULL, NULL,
    avectoroffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, NULL, NULL, NULL, NULL)
```

```
CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, NULL, NULL, alateraloffsetexpression, NULL)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, NULL, NULL, NULL, averticaloffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, NULL, NULL, alateraloffsetexpression,
    averticaloffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, NULL, NULL, avectoroffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, atowardsreferentfeatureID, atowardsreferentname,
   NULL, NULL)
```

```
CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, atowardsreferentfeatureID, atowardsreferentname,
    alateraloffsetexpression, NULL)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN NEW ST_DistanceExp(adistancealong, afromreferentfeatureID,
    afromreferentname, atowardsreferentfeatureID, atowardsreferentname,
    NULL, averticaloffsetexpression)

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    alateraloffsetexpression ST_LatOffsetExp,
    averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_DistanceExp
   (adistancealong ST_LRMeasure,
    afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    atowardsreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength),
    atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.

2) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

3) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.

Description

1) The method *ST_DistanceExp(CHARACTER LARGE OBJECT)* takes the following input parameter:
   a) a CHARACTER LARGE OBJECT value *awktorgml*.
2) For the null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(CHARACTER LARGE OBJECT)*:

   Case:
   a) If *awktorgml* contains a DistanceExpression XML element in the GML representation, then return the result of the value expression: *ST_DisExpFromGML(awktorgml)*.
   b) Otherwise, return the result of the value expression: *ST_DisExpFromText(awktorgml)*.
3) The method *ST_DistanceExp(ST_LRMeasure)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong.*
4) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, NULL, NULL, NULL, NULL, NULL, NULL).*
5) The method *ST_DistanceExp(ST_LRMeasure, ST_LatOffsetExp)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong,*
   b) an *ST_LatOffsetExp* value *alateraloffsetexpression.*
6) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, ST_LatOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, NULL, NULL, NULL, NULL, alateraloffsetexpression, NULL).*
7) The method *ST_DistanceExp(ST_LRMeasure, ST_VerOffsetExp)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong,*
   b) an *ST_VerOffsetExp* value *averticaloffsetexpression.*
8) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, ST_VerOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, NULL, NULL, NULL, NULL, NULL, averticaloffsetexpression).*
9) The method *ST_DistanceExp(ST_LRMeasure, ST_LatOffsetExp, ST_VerOffsetExp)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong,*
   b) an *ST_LatOffsetExp* value *alateraloffsetexpression,*
   c) an *ST_VerOffsetExp* value *averticaloffsetexpression.*
10) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, ST_LatOffsetExp, ST_VerOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, NULL, NULL, NULL, NULL, alateraloffsetexpression, averticaloffsetexpression).*
11) The method *ST_DistanceExp(ST_LRMeasure, ST_VectorOffsetExp)* takes the following input parameters:
    a) an *ST_LRMeasure* value *adistancealong,*
    b) an *ST_VectorOffsetExp* value *avectoroffsetexpression.*
12) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, ST_VectorOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, NULL, NULL, NULL, NULL, avectoroffsetexpression).*

13) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

   a) an *ST_LRMeasure* value *adistancealong,*

   b) a CHARACTER VARYING value *afromreferentfeatureID,*

   c) a CHARACTER VARYING value *afromreferentname.*

14) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, NULL, NULL, NULL, NULL).*

15) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp)* takes the following input parameters:

   a) an *ST_LRMeasure* value *adistancealong,*

   b) a CHARACTER VARYING value *afromreferentfeatureID,*

   c) a CHARACTER VARYING value *afromreferentname,*

   d) an *ST_LatOffsetExp* value *alateraloffsetexpression.*

16) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, NULL, NULL, alateraloffsetexpression, NULL).*

17) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_VerOffsetExp)* takes the following input parameters:

   a) an *ST_LRMeasure* value *adistancealong,*

   b) a CHARACTER VARYING value *afromreferentfeatureID,*

   c) a CHARACTER VARYING value *afromreferentname,*

   d) an *ST_VerOffsetExp* value *averticaloffsetexpression.*

18) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_VerOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, NULL, NULL, NULL, averticaloffsetexpression).*

19) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp, ST_VerOffsetExp)* takes the following input parameters:

   a) an *ST_LRMeasure* value *adistancealong,*

   b) a CHARACTER VARYING value *afromreferentfeatureID,*

   c) a CHARACTER VARYING value *afromreferentname,*

   d) an *ST_LatOffsetExp* value *alateraloffsetexpression,*

   e) an *ST_VerOffsetExp* value *averticaloffsetexpression.*

20) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, INTEGER, ST_LatOffsetExp, ST_VerOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, NULL, NULL, alateraloffsetexpression, averticaloffsetexpression).*

21) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_VectorOffsetExp)* takes the following input parameters:

   a) an *ST_LRMeasure* value *adistancealong,*

   b) a CHARACTER VARYING value *afromreferentfeatureID,*

   c) a CHARACTER VARYING value *afromreferentname,*

   d) an *ST_VectorOffsetExp* value *avectoroffsetexpression.*

22) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, ST_VectorOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, NULL, NULL, avectoroffsetexpression).*

23) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong,*
   b) a CHARACTER VARYING value *afromreferentfeatureID,*
   c) a CHARACTER VARYING value *afromreferentname,*
   d) a CHARACTER VARYING value *atowardsreferentfeatureID,*
   e) an INTEGER ARRAY value *atowardsreferentname.*

24) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, atowardsreferentfeatureID, atowardsreferentname, NULL, NULL).*

25) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong,*
   b) a CHARACTER VARYING value *afromreferentfeatureID,*
   c) a CHARACTER VARYING value *afromreferentname,*
   d) a CHARACTER VARYING value *atowardsreferentfeatureID,*
   e) a CHARACTER VARYING value *atowardsreferentname,*
   f) an *ST_LatOffsetExp* value *alateraloffsetexpression.*

26) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, atowardsreferentfeatureID, atowardsreferentname, alateraloffsetexpression, NULL).*

27) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_VerOffsetExp)* takes the following input parameters:
   a) an *ST_LRMeasure* value *adistancealong,*
   b) a CHARACTER VARYING value *afromreferentfeatureID,*
   c) a CHARACTER VARYING value *afromreferentname,*
   d) a CHARACTER VARYING value *atowardsreferentfeatureID,*
   e) a CHARACTER VARYING value *atowardsreferentname,*
   f) an *ST_VerOffsetExp* value *averticaloffsetexpression.*

28) The null-call type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_VerOffsetExp)* returns the result of the value expression: *NEW ST_LRMeasure(adistancealong, afromreferentfeatureID, afromreferentname, atowardsreferentfeatureID, atowardsreferentname, NULL, averticaloffsetexpression).*

29) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp, ST_VerOffsetExp)* takes the following input parameters:

a) an *ST_LRMeasure* value *adistancealong,*

b) a CHARACTER VARYING value *afromreferentfeatureID,*

c) a CHARACTER VARYING value *afromreferentname,*

d) a CHARACTER VARYING value *atowardsreferentfeatureID,*

e) a CHARACTER VARYING value *atowardsreferentname,*

f) an *ST_LatOffsetExp* value *alateraloffsetexpression,*

g) an *ST_VerOffsetExp* value *averticaloffsetexpression.*

30) The type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_LatOffsetExp, ST_VerOffsetExp)*:

Case:

a) If *adistancealong* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If *atowardsreferentname* is not the null value and *afromreferentname is the null value*, then an exception condition is raised: *SQL/MM Spatial exception – towards referent requires a from referent.*

c) If SELF is the null value, then return the null value.

d) Otherwise, return an *ST_DistanceExp* value with:

i) The *ST_PrivateDistanceAlong* attribute set to *adistancealong.*

ii) The *ST_PrivateFromReferentFeatureID* attribute set to *afromreferentfeatureID.*

iii) The *ST_PrivateFromReferentName* attribute set to *afromreferentname.*

iv) The *ST_PrivateTowardsReferentFeatureID* attribute set to *atowardsreferentfeatureID.*

v) The *ST_PrivateTowardsReferentName* attribute set to *atowardsreferentname.*

vi) The *ST_PrivateLateralOffsetExpression* attribute set to *alateraloffsetexpression.*

vii) The *ST_PrivateVerticalOffsetExpression* attribute set to *averticaloffsetexpression.*

31) The method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_VectorOffsetExp)* takes the following input parameters:

a) an *ST_LRMeasure* value *adistancealong,*

b) a CHARACTER VARYING value *afromreferentfeatureID,*

c) a CHARACTER VARYING value *afromreferentname,*

d) a CHARACTER VARYING value *atowardsreferentfeatureID,*

e) a CHARACTER VARYING value *atowardsreferentname,*

f) an *ST_VectorOffsetExp* value *avectoroffsetexpression.*

32) The type-preserving SQL-invoked constructor method *ST_DistanceExp(ST_LRMeasure, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, CHARACTER VARYING, ST_VectorOffsetExp)*:

Case:

a) If *adistancealong* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If SELF is the null value, then return the null value.

c) Otherwise, return an *ST_DistanceExp* value with:

i) The *ST_PrivateDistanceAlong* attribute set to *adistancealong.*

ii) The *ST_PrivateFromReferentFeatureID* attribute set to *afromreferentfeatureID.*

iii) The *ST_PrivateFromReferentName* attribute set to *afromreferentname.*

iv) The *ST_PrivateTowardsReferentFeatureID* attribute set to *atowardsreferentfeatureID.*

v) The *ST_PrivateTowardsReferentName* attribute set to *atowardsreferentname.*

vi) The *ST_PrivateVectorOffsetExpression* attribute set to *avectoroffsetexpression.*

15.9.3 ST_DistanceAlong Methods

Purpose

Observe and mutate the attribute ST_PrivateDistanceAlong of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_DistanceAlong()
   RETURNS ST_LRMeasure
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateDistanceAlong

CREATE METHOD ST_DistanceAlong
   (adistancealong ST_LRMeasure)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      IF adistancealong IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateDistanceAlong(adistancealong)
            END;
      END IF;
   END
```

Description

1) The method *ST_DistanceAlong()* has no input parameters.

2) The null-call method *ST_DistanceAlong()* returns the value of the *ST_PrivateDistanceAlong* attribute.

3) The method *ST_DistanceAlong(ST_LRMeasure)* takes the following input parameters:

   a) an *ST_LRMeasure* value *adistancealong*.

4) For the type-preserving method *ST_DistanceAlong(ST_LRMeasure)*:

   Case:

   a) If *adistancealong* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateDistanceAlong* set to *adistancealong*.

15.9.4 ST_FromRefFeaID Methods

Purpose

Observe and mutate the attribute ST_PrivateFromReferentFeatureID of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_FromRefFeaID()
   RETURNS CHARACTER VARYING(ST_MaxFeatureIDLength)
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateFromReferentFeatureID

CREATE METHOD ST_FromRefFeaID
   (afromreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateFromReferentFeatureID
             (afromreferentfeatureID)
      END
```

Definitional Rules

1) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

Description

1) The method *ST_FromRefFeaID()* has no input parameters.
2) The null-call method *ST_FromRefFeaID()* returns the value of the *ST_PrivateFromReferentFeatureID* attribute.
3) The method *ST_FromRefFeaID(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *afromreferentfeatureID*.
4) For the type-preserving method *ST_FromRefFeaID(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateFromReferentFeatureID* set to *afromreferentfeatureID*.

15.9.5 ST_FromRefName Methods

Purpose

Observe and mutate the attribute ST_PrivateFromReferentName of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_FromRefName()
   RETURNS CHARACTER VARYING(ST_MaxReferentNameLength)
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateFromReferentName

CREATE METHOD ST_FromRefName
   (afromreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateFromReferentName(afromreferentname)
      END
```

Definitional Rules

1) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.

Description

1) The method *ST_FromRefName()* has no input parameters.

2) The null-call method *ST_FromRefName()* returns the value of the *ST_PrivateFromReferentName* attribute.

3) The method *ST_FromRefName(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *afromreferentname*.

4) For the type-preserving method *ST_FromRefName(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateFromReferentName* set to *afromreferentname*.

15.9.6 ST_TowardsRefFeaID Methods

Purpose

Observe and mutate the attribute ST_PrivateTowardsReferentFeatureID of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_TowardsRefFeaID()
   RETURNS CHARACTER VARYING(ST_MaxFeatureIDLength)
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateTowardsReferentFeatureID

CREATE METHOD ST_TowardsRefFeaID
   (atowardsreferentfeatureID CHARACTER VARYING(ST_MaxFeatureIDLength))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateTowardsReferentFeatureID
             (atowardsreferentfeatureID)
      END
```

Definitional Rules

1) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

Description

1) The method *ST_TowardsRefFeaID()* has no input parameters.
2) The null-call method *ST_TowardsRefFeaID()* returns the value of the *ST_PrivateTowardsReferentFeatureID* attribute.
3) The method *ST_TowardsRefFeaID(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *atowardsreferentfeatureID*.
4) For the type-preserving method *ST_TowardsRefFeaID(CHARACTER VARYING)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateTowardsReferentFeatureID* set to *atowardsreferentfeatureID*.

15.9.7 ST_TowardsRefName Methods

Purpose

Observe and mutate the attribute ST_PrivateTowardsReferentName of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_TowardsRefName()
   RETURNS CHARACTER VARYING(ST_MaxReferentNameLength)
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateTowardsReferentName

CREATE METHOD ST_TowardsRefName
   (atowardsreferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      IF atowardsreferentname IS NOT NULL AND
        SELF.ST_PrivateFromReferentName IS NULL THEN
         SIGNAL SQLSTATE '2FF90'
            SET MESSAGE_TEXT = 'towards referent requires a from
              referent';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateTowardsReferentName(atowardsreferentname)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.

Description

1) The method *ST_TowardsRefName()* has no input parameters.

2) The null-call method *ST_TowardsRefName()* returns the value of the *ST_PrivateTowardsReferentName* attribute.

3) The method *ST_TowardsRefName(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *atowardsreferentname*.

4) For the type-preserving method *ST_TowardsRefName(CHARACTER VARYING)*:

   Case:

   a) If *atowardsreferentname* is not the null value and *SELF. ST_PrivateFromReferentName* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – towards referent requires a from referent*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateTowardsReferentName* set to *atowardsreferentname*.

15.9.8 ST_LatOffsetExp Methods

Purpose

Observe and mutate the attribute ST_PrivateLateralOffsetExpression of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_LatOffsetExp()
   RETURNS ST_LatOffsetExp
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateLateralOffsetExpression

CREATE METHOD ST_LatOffsetExp
   (alateraloffsetexpression ST_LatOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      IF SELF.ST_PrivateVectorOffsetExpression IS
       NOT NULL THEN
         SIGNAL SQLSTATE '2FF91'
            SET MESSAGE_TEXT = 'illegal with vector offset';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateLateralOffsetExpression
                   (alateraloffsetexpression)
            END;
      END IF;
   END
```

Description

1) The method *ST_LatOffsetExp()* has no input parameters.

2) The null-call method *ST_LatOffsetExp()* returns the value of the *ST_PrivateLateralOffsetExpression* attribute.

3) The method *ST_LatOffsetExp(ST_LatOffsetExp)* takes the following input parameters:

   a) an *ST_LatOffsetExp* value *alateraloffsetexpression*.

4) For the type-preserving method *ST_LatOffsetExp(ST_LatOffsetExp)*:

   Case:

: a) If *SELF.ST_PrivateVectorOffsetExpression* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with vector offset*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateLateralOffsetExpression* set to *alateraloffsetexpression*.

15.9.9 ST_VerOffsetExp Methods

Purpose

Observe and mutate the attribute ST_PrivateVerticalOffsetExpression of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_VerOffsetExp()
   RETURNS ST_VerOffsetExp
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateVerticalOffsetExpression

CREATE METHOD ST_VerOffsetExp
   (averticaloffsetexpression ST_VerOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      IF SELF.ST_PrivateVectorOffsetExpression IS
       NOT NULL THEN
         SIGNAL SQLSTATE '2FF91'
            SET MESSAGE_TEXT = 'illegal with vector offset';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateVerticalOffsetExpression
                   (averticaloffsetexpression)
            END;
      END IF;
   END
```

Description

1) The method *ST_VerOffsetExp()* has no input parameters.

2) The null-call method *ST_VerOffsetExp()* returns the value of the *ST_PrivateVerticalOffsetExpression* attribute.

3) The method *ST_VerOffsetExp(ST_VerOffsetExp)* takes the following input parameters:

   a) an *ST_VerOffsetExp* value *averticaloffsetexpression*.

4) For the type-preserving method *ST_VerOffsetExp(ST_VerOffsetExp)*:

   Case:

   a) If *SELF.ST_PrivateVectorOffsetExpression* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with vector offset*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateVerticalOffsetExpression* set to *averticaloffsetexpression*.

15.9.10 ST_VectorOffsetExp Methods

Purpose

Observe and mutate the attribute ST_PrivateVectorOffsetExpression of an ST_DistanceExp value.

Definition

```
CREATE METHOD ST_VectorOffsetExp()
   RETURNS ST_VectorOffsetExp
   FOR ST_DistanceExp
   RETURN SELF.ST_PrivateVectorOffsetExpression

CREATE METHOD ST_VectorOffsetExp
   (avectoroffsetexpression ST_VectorOffsetExp)
   RETURNS ST_DistanceExp
   FOR ST_DistanceExp
   BEGIN
      IF SELF.ST_PrivateLateralOffsetExpression IS
       NOT NULL THEN
         SIGNAL SQLSTATE '2FF92'
            SET MESSAGE_TEXT = 'illegal with lateral offset';
      ELSEIF
       SELF.ST_PrivateVerticalOffsetExpression IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF93'
            SET MESSAGE_TEXT = 'illegal with vertical offset';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateVectorOffsetExpression
                   (avectoroffsetexpression)
            END;
      END IF;
   END
```

Description

1) The method *ST_VectorOffsetExp()* has no input parameters.

2) The null-call method *ST_VectorOffsetExp()* returns the value of the *ST_PrivateVectorOffsetExpression* attribute.

3) The method *ST_VectorOffsetExp(ST_VectorOffsetExp)* takes the following input parameters:

   a) an *ST_VectorOffsetExp* value *avectoroffsetexpression*.

4) For the type-preserving method *ST_VectorOffsetExp(ST_VectorOffsetExp)*:

   Case:

   a) If *SELF.ST_PrivateLateralOffsetExpression* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with lateral offset*.

   b) If *SELF.ST_PrivateVerticalOffsetExpression* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with vertical offset*.

   c) If SELF is the null value, then return the null value.

   d) Otherwise, return an *ST_DistanceExp* value with the attribute *ST_PrivateVectorOffsetExpression* set to *avectoroffsetexpression*.

15.9.11 ST_DisExpFromText Function

Purpose

Return an ST_DistanceExp value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_DistanceExp value.

Definition

```
CREATE FUNCTION ST_DisExpFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxLRAsText))
   RETURNS ST_DistanceExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of a linear referencing type value.

Description

1) The function *ST_DisExpFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) For the null-call function *ST_DisExpFromText(CHARACTER LARGE OBJECT)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_DistanceExp* value.

      If *awkt* is not producible in the BNF for <distance expression text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_DisExpFromText(awkt) AS ST_DistanceExp)*.

15.9.12 ST_DisExpFromGML Function

Purpose

Return an ST_DistanceExp value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Distance Expression representation of an ST_DistanceExp value.

Definition

```
CREATE FUNCTION ST_DisExpFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxLRAsGML))
   RETURNS ST_DistanceExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

Description

1) The function *ST_DisExpFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_DisExpFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain a DistanceExpression XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_DisExpFromGML(agml) AS ST_DistanceExp)*.

15.10 ST_Referent Type and Routines

15.10.1 ST_Referent Type

Purpose

The ST_Referent type specifies a known location along an owning ST_LRFeature. The ST_Referent type is instantiable.

Definition

```
CREATE TYPE ST_Referent
   AS (
      ST_PrivateReferentName CHARACTER VARYING(ST_MaxReferentNameLength)
       DEFAULT NULL,
      ST_PrivateReferentType CHARACTER VARYING(30) DEFAULT NULL,
      ST_PrivatePosition ST_Point DEFAULT NULL,
      ST_PrivateLocation ST_PositionExp DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Referent
      (areferentname CHARACTER VARYING(MaxReferentNameLength),
       areferenttype CHARACTER VARYING(30))
      RETURNS ST_Referent
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Referent
      (areferentname CHARACTER VARYING(MaxReferentNameLength),
       areferenttype CHARACTER VARYING(30),
       aposition ST_Point)
      RETURNS ST_Referent
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Referent
      (areferentname CHARACTER VARYING(MaxReferentNameLength),
       areferenttype CHARACTER VARYING(30),
       alocation ST_PositionExp)
      RETURNS ST_Referent
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Referent
   (areferentname CHARACTER VARYING(MaxReferentNameLength),
    areferenttype CHARACTER VARYING(30),
    aposition ST_Point,
    alocation ST_PositionExp)
   RETURNS ST_Referent
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ReferentName()
   RETURNS CHARACTER VARYING(MaxReferentNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ReferentName
   (areferentname CHARACTER VARYING(MaxReferentNameLength))
   RETURNS ST_Referent
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ReferentType()
   RETURNS CHARACTER VARYING(30)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ReferentType
   (areferenttype CHARACTER VARYING(30))
   RETURNS ST_Referent
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Position()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Position
   (aposition ST_Point)
   RETURNS ST_Referent
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_Location()
   RETURNS ST_PositionExp
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Location
   (alocation ST_PositionExp)
   RETURNS ST_Referent
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ChangePosAndLoc
   (aposition ST_Point,
    alocation ST_PositionExp)
   RETURNS ST_Referent
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.
2) The attribute *ST_PrivateReferentName* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferentName*.
3) The attribute *ST_PrivateReferentType* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferentType.*
4) The attribute *ST_PrivatePosition* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivatePosition*.
5) The attribute *ST_PrivateLocation* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLocation*.

Description

1) The *ST_Referent* type provides for public use:
   a) a method *ST_Referent(CHARACTER VARYING*, *CHARACTER VARYING)*,
   b) a method *ST_Referent(CHARACTER VARYING*, *CHARACTER VARYING, ST_Point)*,
   c) a method *ST_Referent(CHARACTER VARYING*, *CHARACTER VARYING, ST_PositionExp)*,
   d) a method *ST_Referent(CHARACTER VARYING*, *CHARACTER VARYING, ST_Point, ST_PositionExp)*,
   e) a method *ST_ReferentName()*,
   f) a method *ST_ReferentName(CHARACTER VARYING)*,
   g) a method *ST_ReferentType()*,
   h) a method *ST_ReferentType(CHARACTER VARYING)*,
   i) a method *ST_Position()*,
   j) a method *ST_Position(ST_Point)*,
   k) a method *ST_Location()*,

l) a method *ST_Location(ST_PositionExp),*

m) a method *ST_ChangePosAndLoc(ST_Point, ST_PositionExp).*

2) The *ST_PrivateReferentName* attribute contains the CHARACTER VARYING referent name value.

3) The *ST_PrivateReferentType* attribute contains the CHARACTER VARYING referent type value.

4) The *ST_PrivatePosition* attribute contains the optional *ST_Point* referent position value.

5) The *ST_PrivateLocation* attribute contains the optional *ST_PositionExp* referent location value.

6) An *ST_Referent* type value cannot exist in isolation – it must be owned by an *ST_LRFeature* type value and be a member of that *ST_LRFeature*'s *ST_PrivateReferents* collection of *ST_Referent* values.

7) The *ST_PrivateReferentName* attribute value shall be unique across all *ST_Referent* values contained in an *ST_LRFeature ST_PrivateReferents* attribute collection of *ST_Referent* values.

8) The allowable referent type values specified by the *ST_PrivateReferentType* attribute shall include 'reference marker', 'intersection', 'boundary' and 'landmark'.

9) At least one of the attributes *ST_PrivatePosition* and *ST_PrivateLocation* is usually not NULL unless the location of the referent can be implied. For example, if the Linear Referencing Method is of type 'mile marker', then a *St_PrivateReferentName* attribute having a value of '1' would represent a location which is one mile along the *ST_LRFeature*.

10) If both of the attributes *ST_PrivatePosition* and *ST_PrivateLocation* are not NULL, then they shall both refer to the same physical location.

11) The *ST_DistanceExp* of the *ST_PrivateLocation* attribute *ST_PositionExp* value shall not contain an offset expression; all referents shall lie on the *ST_LRFeature*.

12) Let *RPE* be the *SELF.ST_PrivateLocation* attribute *ST_PositionExp* value. Let *RLRM* be the *RPE.ST_PrivateLRMID* attribute INTEGER value. Let *PE* be any *ST_PositionExp* value having an *ST_PrivateDistanceExpression* attribute *ST_DistanceExp* value which uses SELF as a "from" or "towards" referent. Let *PLRM* be a *PE.ST_PrivateLRMID* attribute INTEGER value. There is no requirement that *RLRM = PLRM*.

15.10.2 ST_Referent Methods

Purpose

Return an ST_Referent value constructed from either:

a) the specified CHARACTER VARYING referent name and CHARACTER VARYING referent type values;

b) the specified CHARACTER VARYING referent name, CHARACTER VARYING referent type and ST_Point referent position values;

c) the specified CHARACTER VARYING referent name, CHARACTER VARYING referent type and ST_PositionExp referent location values;

d) the specified CHARACTER VARYING referent name, CHARACTER VARYING referent type, ST_Point referent position and ST_PositionExp referent location values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Referent
   (areferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    areferenttype CHARACTER VARYING(30))
   RETURNS ST_Referent
   FOR ST_Referent
   RETURN NEW ST_Referent(areferentname, areferenttype, NULL, NULL)

CREATE CONSTRUCTOR METHOD ST_Referent
   (areferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    areferenttype CHARACTER VARYING(30),
    aposition ST_Point)
   RETURNS ST_Referent
   FOR ST_Referent
   RETURN NEW ST_Referent(areferentname, areferenttype, aposition, NULL)

CREATE CONSTRUCTOR METHOD ST_Referent
   (areferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    areferenttype CHARACTER VARYING(30),
    alocation ST_PositionExp)
   RETURNS ST_Referent
   FOR ST_Referent
   RETURN NEW ST_Referent(areferentname, areferenttype, NULL, alocation)

CREATE CONSTRUCTOR METHOD ST_Referent
   (areferentname CHARACTER VARYING(ST_MaxReferentNameLength),
    areferenttype CHARACTER VARYING(30),
    aposition ST_Point,
    alocation ST_PositionExp)
   RETURNS ST_Referent
   FOR ST_Referent
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.

Description

1) The method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *areferentname,*

b) a CHARACTER VARYING value *areferenttype.*

2) The null-call type-preserving SQL-invoked constructor method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING)* returns the result of the value expression: *NEW ST_Referent(areferentname, areferenttype, NULL, NULL).*

3) The method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING, ST_Point)* takes the following input parameters:

   a) a CHARACTER VARYING value *areferentname,*

   b) a CHARACTER VARYING value *areferenttype,*

   c) an *ST_Point* value *aposition.*

4) The null-call type-preserving SQL-invoked constructor method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING, ST_Point)* returns the result of the value expression: *NEW ST_Referent(areferentname, areferenttype, aposition, NULL).*

5) The method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING, ST_PositionExp)* takes the following input parameters:

   a) a CHARACTER VARYING value *areferentname,*

   b) a CHARACTER VARYING value *areferenttype,*

   c) an *ST_PositionExp* value *alocation.*

6) The null-call type-preserving SQL-invoked constructor method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING, ST_PositionExp)* returns the result of the value expression: *NEW ST_Referent(areferentname, areferenttype, NULL, alocation).*

7) The method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING, ST_Point, ST_PositionExp)* takes the following input parameters:

   a) a CHARACTER VARYING value *areferentname,*

   b) a CHARACTER VARYING value *areferenttype,*

   c) an *ST_Point* value *aposition,*

   d) an *ST_PositionExp* value *alocation.*

8) For the null-call type-preserving SQL-invoked constructor method *ST_Referent(CHARACTER VARYING, CHARACTER VARYING, ST_Point, ST_PositionExp)*:

   a) If *areferentname* is the null value or if *areferenttype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *aposition* and *alocation* are both NOT NULL and if either they represent different physical locations or if it is not possible to determine if they represent the same physical location, then an exception condition is raised: *SQL/MM Spatial exception – potentially incompatible referent position and location*.

   c) Otherwise, return an *ST_Referent* value with:

      i) The *ST_PrivateReferentName* attribute set to *areferentname.*

      ii) The *ST_PrivateReferentType* attribute set to *areferenttype.*

      iii) The *ST_PrivatePosition* attribute set to *aposition.*

      iv) The *ST_PrivateLocation* attribute set to *alocation.*

15.10.3 ST_ReferentName Methods

Purpose

Observe and mutate the attribute ST_PrivateReferentName of an ST_Referent value.

Definition

```
CREATE METHOD ST_ReferentName()
   RETURNS CHARACTER VARYING(ST_MaxReferentNameLength)
   FOR ST_Referent
   RETURN SELF.ST_PrivateReferentName

CREATE METHOD ST_ReferentName
   (areferentname CHARACTER VARYING(ST_MaxReferentNameLength))
   RETURNS ST_Referent
   FOR ST_Referent
   BEGIN
      IF areferentname IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateReferentName(areferentname)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxReferentNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a referent.

Description

1) The method *ST_ReferentName()* has no input parameters.
2) The null-call method *ST_ReferentName()* returns the value of the *ST_PrivateReferentName* attribute.
3) The method *ST_ReferentName(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *areferentname*.
4) For the type-preserving method *ST_ReferentName(CHARACTER VARYING)*:

   Case:

   a) If *areferentname* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_Referent* value with the attribute *ST_PrivateReferentName* set to *areferentname*.

15.10.4 ST_ReferentType Methods

Purpose

Observe and mutate the attribute ST_PrivateReferentType of an ST_Referent value.

Definition

```
CREATE METHOD ST_ReferentType()
   RETURNS CHARACTER VARYING(30)
   FOR ST_Referent
   RETURN SELF.ST_PrivateReferentType

CREATE METHOD ST_ReferentType
   (areferenttype CHARACTER VARYING(30))
   RETURNS ST_Referent
   FOR ST_Referent
   BEGIN
      IF areferenttype IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateReferentType(areferenttype)
            END;
      END IF;
   END
```

Description

1) The method *ST_ReferentType()* has no input parameters.

2) The null-call method *ST_ReferentType()* returns the value of the *ST_PrivateReferentType* attribute.

3) The method *ST_ReferentType(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *areferenttype*.

4) For the type-preserving method *ST_ReferentType(CHARACTER VARYING)*:

   Case:

   a) If *areferenttype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_Referent* value with the attribute *ST_PrivateReferentType* set to *areferenttype*.

15.10.5 ST_Position Methods

Purpose

Observe and mutate the attribute ST_PrivatePosition of an ST_Referent value.

Definition

```
CREATE METHOD ST_Position()
   RETURNS ST_Point
   FOR ST_Referent
   RETURN SELF.ST_PrivatePosition

CREATE METHOD ST_Position
   (aposition ST_Point)
   RETURNS ST_Referent
   FOR ST_Referent
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_Position()* has no input parameters.

2) The null-call method *ST_Position()* returns the value of the *ST_PrivatePosition* attribute.

3) The method *ST_Position(ST_Point)* takes the following input parameters:

   a) an *ST_Point* value *aposition*.

4) For the type-preserving method *ST_Position(ST_Point)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise,

      Case:

      i) If *aposition* and *SELF.ST_Location()* are both NOT NULL and if either they represent different physical locations or if it is not possible to determine if they represent the same physical location, then an exception condition is raised: *SQL/MM Spatial exception – potentially incompatible referent position and location*.

      ii) Otherwise, return an *ST_Referent* value with the attribute *ST_PrivatePosition* set to *aposition*.

15.10.6 ST_Location Methods

Purpose

Observe and mutate the attribute ST_PrivateLocation of an ST_Referent value.

Definition

```
CREATE METHOD ST_Location()
   RETURNS ST_PositionExp
   FOR ST_Referent
   RETURN SELF.ST_PrivateLocation

CREATE METHOD ST_Location
   (alocation ST_PositionExp)
   RETURNS ST_Referent
   FOR ST_Referent
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_Location()* has no input parameters.

2) The null-call method *ST_Location()* returns the value of the *ST_PrivateLocation* attribute.

3) The method *ST_Location(ST_PositionExp)* takes the following input parameters:

   a) an *ST_PositionExp* value *alocation*.

4) For the type-preserving method *ST_Location(ST_PositionExp)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise,

      Case:

      i) If *SELF.ST_Position()* and *alocation* are both NOT NULL and if either they represent different physical locations or if it is not possible to determine if they represent the same physical location, then an exception condition is raised: *SQL/MM Spatial exception – potentially incompatible referent position and location*.

      ii) Otherwise, return an *ST_Referent* value with the attribute *ST_PrivateLocation* set to *alocation*.

15.10.7 ST_ChangePosAndLoc Method

Purpose

Simultaneously mutate the ST_PrivatePosition and ST_PrivateLocation attributes of an ST_Referent value.

Definition

```
CREATE METHOD ST_ChangePosAndLoc
   (aposition ST_Point,
    alocation ST_PositionExp)
   RETURNS ST_Referent
   FOR ST_Referent
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            --
            -- See Description
            --
      END
```

Description

1) The method *ST_ChangePosAndLoc(ST_Point, ST_PositionExp)* takes the following input parameters:

   a) an *ST_Point* value *aposition*,

   b) an *ST_PositionExp* value *alocation*.

2) For the type-preserving method *ST_ChangePosAndLoc(ST_Point, ST_PositionExp)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise,

      i) If *aposition* and *alocation* are both NOT NULL and if either they represent different physical locations or if it is not possible to determine if they represent the same physical location, then an exception condition is raised: *SQL/MM Spatial exception – potentially incompatible referent position and location.*

      ii) Otherwise, return an *ST_Referent* value with:

         1) the attribute *ST_PrivatePosition* set to *aposition*.

         2) the attribute *ST_PrivateLocation* set to *alocation*.

15.11 ST_LatOffsetExp Type and Routines

15.11.1 ST_LatOffsetExp Type

Purpose

The ST_LatOffsetExp type specifies the lateral offset for a linearly referenced location. The ST_LatOffsetExp type is instantiable.

Definition

```
CREATE TYPE ST_LatOffsetExp
   AS (
      ST_PrivateOffsetLateralDistance ST_LRMeasure DEFAULT NULL,
      ST_PrivateFeatureGeometry ST_Geometry DEFAULT NULL,
      ST_PrivateOffsetReferentDescription CHARACTER VARYING(128)
         DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_LatOffsetExp
      (anoffsetlateraldistance ST_LRMeasure)
      RETURNS ST_LatOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LatOffsetExp
      (anoffsetlateraldistance ST_LRMeasure,
       afeaturegeometry ST_Geometry)
      RETURNS ST_LatOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_LatOffsetExp
      (anoffsetlateraldistance ST_LRMeasure
       anoffsetreferentdescription CHARACTER VARYING(128))
      RETURNS ST_LatOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_OffsetLatDist()
      RETURNS ST_LRMeasure
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_OffsetLatDist
   (anoffsetlateraldistance ST_LRMeasure)
   RETURNS ST_LatOffsetExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_FeatureGeometry()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_FeatureGeometry
   (afeaturegeometry ST_Geometry)
   RETURNS ST_LatOffsetExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_OffsetRefDesc()
   RETURNS CHARACTER VARYING(128)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_OffsetRefDesc
   (anoffsetreferentdescription CHARACTER VARYING(128))
   RETURNS ST_LatOffsetExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateOffsetLateralDistance* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateOffsetLateralDistance*.
2) The attribute *ST_PrivateFeatureGeometry* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateFeatureGeometry*.
3) The attribute *ST_PrivateOffsetReferentDescription* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateOffsetReferentDescription*.

Description

1) The *ST_LatOffsetExp* type provides for public use:
   a) a method *ST_LatOffsetExp(ST_LRMeasure)*,
   b) a method *ST_LatOffsetExp(ST_LRMeasure, ST_Geometry)*,
   c) a method *ST_LatOffsetExp(ST_LRMeasure, CHARACTER VARYING)*,
   d) a method *ST_OffsetLatDist()*,
   e) a method *ST_OffsetLatDist(ST_LRMeasure)*,

f) a method *ST_FeatureGeometry()*,

g) a method *ST_FeatureGeometry(ST_Geometry)*,

h) a method *ST_OffsetRefDesc()*,

i) a method *ST_OffsetRefDesc(CHARACTER VARYING).*

2) The *ST_PrivateOffsetLateralDistance* attribute contains the *ST_LRMeasure* offset lateral distance measure value.

3) The *ST_PrivateFeatureGeometry* attribute contains the *ST_Geometry* feature geometry lateral offset referent value.

4) The *ST_PrivateOffsetReferentDescription* attribute contains the CHARACTER VARYING lateral offset referent description value.

5) The optional *ST_PrivateOffsetLateralDistance* attribute value is the distance measured left or right of and perpendicular to the lateral offset referent (or left or right of and perpendicular to the linear element being measured if no lateral offset referent is specified) to the position being specified. A positive (+) value is measured in the direction specified by the *ST_PrivatePositiveLateralOffsetDirection* attribute value of the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* having SELF as its lateral offset expression. A NULL value or a value of 0 (zero) is to be interpreted as not having a lateral displacement from the lateral offset referent (or the linear element being measured if no lateral offset referent is specified).

6) If the optional *ST_PrivateUnits* attribute value of the *ST_LRMeasure* value specified by the *ST_PrivateOffsetLateralDistance* attribute is NULL, then the *ST_PrivateOffsetUnits* attribute value of the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* having SELF as its lateral offset expression shall apply.

7) An optional lateral offset referent is specified by either a geometry of a feature or a lateral offset referent described as a character string (e.g., "back of curb"). Consequently, at least one of *ST_PrivateFeatureGeometry* or *ST_PrivateOffsetReferentDescription* must be NULL.

15.11.2 ST_LatOffsetExp Methods

Purpose

Return an ST_LatOffsetExp value constructed from either:

a) the specified ST_LRMeasure offset lateral distance measure value;

b) the specified ST_LRMeasure offset lateral distance measure and ST_Geometry feature geometry values;

c) the specified ST_LRMeasure offset lateral distance measure and CHARACTER VARYING offset referent description values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_LatOffsetExp
   (anoffsetlateraldistance ST_LRMeasure)
   RETURNS ST_LatOffsetExp
   FOR ST_LatOffsetExp
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE CONSTRUCTOR METHOD ST_LatOffsetExp
   (anoffsetlateraldistance ST_LRMeasure,
    afeaturegeometry ST_Geometry)
   RETURNS ST_LatOffsetExp
   FOR ST_LatOffsetExp
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE CONSTRUCTOR METHOD ST_LatOffsetExp
   (anoffsetlateraldistance ST_LRMeasure,
    anoffsetreferentdescription CHARACTER VARYING(128))
   RETURNS ST_LatOffsetExp
   FOR ST_LatOffsetExp
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_LatOffsetExp(ST_LRMeasure)* takes the following input parameters:

   a) an *ST_LRMeasure* value *anoffsetlateraldistance.*

2) The null-call type-preserving SQL-invoked constructor method *ST_LatOffsetExp(ST_LRMeasure)* returns an *ST_LatOffsetExp* value with:

   a) The *ST_PrivateOffsetLateralDistance* attribute set to *anoffsetlateraldistance.*

3) The method *ST_LatOffsetExp(ST_LRMeasure, ST_Geometry)* takes the following input parameters:

   a) an *ST_LRMeasure* value *anoffsetlateraldistance,*

   b) an *ST_Geometry* value *afeaturegeometry*.

4) The null-call type-preserving SQL-invoked constructor method *ST_LatOffsetExp(ST_LRMeasure, ST_Geometry)* returns an *ST_LatOffsetExp* value with:

   a) The *ST_PrivateOffsetLateralDistance* attribute set to *anoffsetlateraldistance.*

b) The *ST_PrivateFeatureGeometry* attribute set to *afeaturegeometry.*

5) The method *ST_LatOffsetExp(ST_LRMeasure, CHARACTER VARYING)* takes the following input parameters:

a) an *ST_LRMeasure* value *anoffsetlateraldistance,*

b) a CHARACTER VARYING value *anoffsetreferentdescription*.

6) The null-call type-preserving SQL-invoked constructor method *ST_LatOffsetExp(ST_LRMeasure, CHARACTER VARYING)* returns an *ST_LatOffsetExp* value with:

a) The *ST_PrivateOffsetLateralDistance* attribute set to *anoffsetlateraldistance.*

b) The *ST_PrivateOffsetReferentDescription* attribute set to *anoffsetreferentdescription.*

15.11.3 ST_OffsetLatDist Methods

Purpose

Observe and mutate the attribute ST_PrivateOffsetLateralDistance of an ST_LatOffsetExp value.

Definition

```
CREATE METHOD ST_OffsetLatDist()
   RETURNS ST_LRMeasure
   FOR ST_LatOffsetExp
   RETURN SELF.ST_PrivateOffsetLateralDistance

CREATE METHOD ST_OffsetLatDist
   (anoffsetlateraldistance ST_LRMeasure)
   RETURNS ST_LatOffsetExp
   FOR ST_LatOffsetExp
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateOffsetLateralDistance
             (anoffsetlateraldistance)
      END
```

Description

1) The method *ST_OffsetLatDist()* has no input parameters.
2) The null-call method *ST_OffsetLatDist()* returns the value of the *ST_PrivateOffsetLateralDistance* attribute.
3) The method *ST_OffsetLatDist(ST_LRMeasure)* takes the following input parameters:
   a) an *ST_LRMeasure* value *anoffsetlateraldistance*.
4) For the type-preserving method *ST_OffsetLatDist(ST_LRMeasure)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_LatOffsetExp* value with the attribute *ST_PrivateOffsetLateralDistance* set to *anoffsetlateraldistance*.

15.11.4 ST_FeatureGeometry Methods

Purpose

Observe and mutate the attribute ST_PrivateFeatureGeometry of an ST_LatOffsetExp value.

Definition

```
CREATE METHOD ST_FeatureGeometry()
   RETURNS ST_Geometry
   FOR ST_LatOffsetExp
   RETURN SELF.ST_PrivateFeatureGeometry

CREATE METHOD ST_FeatureGeometry
   (afeaturegeometry ST_Geometry)
   RETURNS ST_LatOffsetExp
   FOR ST_LatOffsetExp
   BEGIN
      IF afeaturegeometry IS NOT NULL and
        SELF.ST_PrivateOffsetReferentDescription IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF94'
            SET MESSAGE_TEXT = 'illegal with offset referent description';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateFeatureGeometry
                   (afeaturegeometry)
            END;
      END IF;
   END
```

Description

1) The method *ST_FeatureGeometry()* has no input parameters.

2) The null-call method *ST_FeatureGeometry()* returns the value of the *ST_PrivateFeatureGeometry* attribute.

3) The method *ST_FeatureGeometry(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *afeaturegeometry*.

4) For the type-preserving method *ST_FeatureGeometry(ST_Geometry)*:

   Case:

   a) If *afeaturegeometry* is not the null value and *SELF.ST_PrivateOffsetReferentDescription* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with offset referent description.*

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LatOffsetExp* value with the attribute *ST_PrivateFeatureGeometry* set to *afeaturegeometry*.

15.11.5 ST_OffsetRefDesc Methods

Purpose

Observe and mutate the attribute ST_PrivateOffsetReferentDescription of an ST_LatOffsetExp value.

Definition

```
CREATE METHOD ST_OffsetRefDesc()
   RETURNS CHARACTER VARYING(128)
   FOR ST_LatOffsetExp
   RETURN SELF.ST_PrivateOffsetReferentDescription

CREATE METHOD ST_OffsetRefDesc
   (anoffsetreferentdescription CHARACTER VARYING(128))
   RETURNS ST_LatOffsetExp
   FOR ST_LatOffsetExp
   BEGIN
      IF anoffsetreferentdescription IS NOT NULL and
        SELF.ST_PrivateFeatureGeometry IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF95'
            SET MESSAGE_TEXT = 'illegal with offset referent geometry';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateOffsetReferentDescription
                    (anoffsetreferentdescription)
            END;
      END IF;
   END
```

Description

1) The method *ST_OffsetRefDesc()* has no input parameters.

2) The null-call method *ST_OffsetRefDesc()* returns the value of the *ST_PrivateOffsetReferentDescription* attribute.

3) The method *ST_OffsetRefDesc(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *anoffsetreferentdescription*.

4) For the type-preserving method *ST_OffsetRefDesc(CHARACTER VARYING)*:

   Case:

   a) If *anoffsetreferentdescription* is not the null value and *SELF.ST_PrivateFeatureGeometry* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with offset referent geometry*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_LatOffsetExp* value with the attribute *ST_PrivateOffsetReferentDescription* set to *anoffsetreferentdescription*.

15.12 ST_VerOffsetExp Type and Routines

15.12.1 ST_VerOffsetExp Type

Purpose

The ST_VerOffsetExp type specifies the vertical offset for a linearly referenced location. The ST_VerOffsetExp type is instantiable.

Definition

```
CREATE TYPE ST_VerOffsetExp
   AS (
      ST_PrivateOffsetVerticalDistance ST_LRMeasure DEFAULT NULL,
      ST_PrivateFeatureGeometry ST_Geometry DEFAULT NULL,
      ST_PrivateOffsetReferentDescription CHARACTER VARYING(128)
         DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_VerOffsetExp
      (anoffsetverticaldistance ST_LRMeasure)
      RETURNS ST_VerOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_VerOffsetExp
      (anoffsetverticaldistance ST_LRMeasure,
       afeaturegeometry ST_Geometry)
      RETURNS ST_VerOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_VerOffsetExp
      (anoffsetverticaldistance ST_LRMeasure
       anoffsetreferentdescription CHARACTER VARYING(128))
      RETURNS ST_VerOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_OffsetVerDist()
      RETURNS ST_LRMeasure
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_OffsetVerDist
   (anoffsetverticaldistance ST_LRMeasure)
   RETURNS ST_VerOffsetExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_FeatureGeometry()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_FeatureGeometry
   (afeaturegeometry ST_Geometry)
   RETURNS ST_VerOffsetExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_OffsetRefDesc()
   RETURNS CHARACTER VARYING(128)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_OffsetRefDesc
   (anoffsetreferentdescription CHARACTER VARYING(128))
   RETURNS ST_VerOffsetExp
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateOffsetVerticalDistance* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateOffsetVerticalDistance*.
2) The attribute *ST_PrivateFeatureGeometry* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateFeatureGeometry*.
3) The attribute *ST_PrivateOffsetReferentDescription* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateOffsetReferentDescription*.

Description

1) The *ST_VerOffsetExp* type provides for public use:
   a) a method *ST_VerOffsetExp(ST_LRMeasure)*,
   b) a method *ST_VerOffsetExp(ST_LRMeasure, ST_Geometry)*,
   c) a method *ST_VerOffsetExp(ST_LRMeasure, CHARACTER VARYING)*,
   d) a method *ST_OffsetVerDist()*,

e) a method *ST_OffsetVerDist(ST_LRMeasure)*,

f) a method *ST_FeatureGeometry()*,

g) a method *ST_FeatureGeometry(ST_Geometry)*,

h) a method *ST_OffsetRefDesc()*,

i) a method *ST_OffsetRefDesc(CHARACTER VARYING).*

2) The *ST_PrivateOffsetVerticalDistance* attribute contains the *ST_LRMeasure* offset vertical distance measure value.

3) The *ST_PrivateFeatureGeometry* attribute contains the *ST_Geometry* feature geometry vertical offset referent value.

4) The *ST_PrivateOffsetReferentDescription* attribute contains the CHARACTER VARYING vertical offset referent description value.

5) The optional *ST_PrivateOffsetVerticalDistance* attribute value is the measure of the vertical offset of the distance expression. This is the distance above or below the vertical offset referent (or, if no vertical offset referent is specified, then above or below the lateral offset referent if one is specified; otherwise, above or below the linear element being measured) to the position being specified. A positive (+) value is measured in the direction specified by the *ST_PrivatePositiveVerticalOffsetDirection* attribute value of the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* having SELF as its vertical offset expression. A NULL value or a value of 0 (zero) is to be interpreted as not having a vertical displacement.

6) If the optional *ST_PrivateUnits* attribute value of the *ST_LRMeasure* value specified by the *ST_PrivateOffsetVerticalDistance* attribute is NULL, then the *ST_PrivateOffsetUnits* attribute value of the *ST_LRM* Linear Referencing Method associated with the *ST_DistanceExp* having SELF as its vertical offset expression shall apply.

7) An optional vertical offset referent is specified by either a geometry of a feature or a vertical offset referent described as a character string (e.g., "existing ground at lateral offset"). Consequently, at least one of *ST_PrivateFeatureGeometry* or *ST_PrivateOffsetReferentDescription* must be NULL.

15.12.2 ST_VerOffsetExp Methods

Purpose

Return an ST_VerOffsetExp value constructed from either:

a) the specified ST_LRMeasure offset vertical distance measure value;

b) the specified ST_LRMeasure offset vertical distance measure and ST_Geometry feature geometry values;

c) the specified ST_LRMeasure offset vertical distance measure and CHARACTER VARYING offset referent description values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_VerOffsetExp
   (anoffsetverticaldistance ST_LRMeasure)
   RETURNS ST_VerOffsetExp
   FOR ST_VerOffsetExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_VerOffsetExp
   (anoffsetverticaldistance ST_LRMeasure,
    afeaturegeometry ST_Geometry)
   RETURNS ST_VerOffsetExp
   FOR ST_VerOffsetExp
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_VerOffsetExp
   (anoffsetverticaldistance ST_LRMeasure,
    anoffsetreferentdescription CHARACTER VARYING(128))
   RETURNS ST_VerOffsetExp
   FOR ST_VerOffsetExp
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_VerOffsetExp(ST_LRMeasure)* takes the following input parameters:

   a) an *ST_LRMeasure* value *anoffsetverticaldistance.*

2) The null-call type-preserving SQL-invoked constructor method *ST_VerOffsetExp(ST_LRMeasure)* returns an *ST_VerOffsetExp* value with:

   a) The *ST_PrivateOffsetVerticalDistance* attribute set to *anoffsetverticaldistance.*

3) The method *ST_VerOffsetExp(ST_LRMeasure, ST_Geometry)* takes the following input parameters:

   a) an *ST_LRMeasure* value *anoffsetverticaldistance,*

   b) an *ST_Geometry* value *afeaturegeometry*.

4) The null-call type-preserving SQL-invoked constructor method *ST_VerOffsetExp(ST_LRMeasure, ST_Geometry)* returns an *ST_VerOffsetExp* value with:

   a) The *ST_PrivateOffsetVerticalDistance* attribute set to *anoffsetverticaldistance.*

   b) The *ST_PrivateFeatureGeometry* attribute set to *afeaturegeometry.*

5) The method *ST_VerOffsetExp(ST_LRMeasure, CHARACTER VARYING)* takes the following input parameters:
   a) an *ST_LRMeasure* value *anoffsetverticaldistance,*
   b) a CHARACTER VARYING value *anoffsetreferentdescription*.
6) The null-call type-preserving SQL-invoked constructor method *ST_VerOffsetExp(ST_LRMeasure, CHARACTER VARYING)* returns an *ST_VerOffsetExp* value with:
   a) The *ST_PrivateOffsetVerticalDistance* attribute set to *anoffsetverticaldistance.*
   b) The *ST_PrivateOffsetReferentDescription* attribute set to *anoffsetreferentdescription.*

15.12.3 ST_OffsetVerDist Methods

Purpose

Observe and mutate the attribute ST_PrivateOffsetVerticalDistance of an ST_VerOffsetExp value.

Definition

```
CREATE METHOD ST_OffsetVerDist()
   RETURNS ST_LRMeasure
   FOR ST_VerOffsetExp
   RETURN SELF.ST_PrivateOffsetVerticalDistance

CREATE METHOD ST_OffsetVerDist
   (anoffsetverticaldistance ST_LRMeasure)
   RETURNS ST_VerOffsetExp
   FOR ST_VerOffsetExp
   RETURN
      CASE
         WHEN SELF IS NULL THEN
            NULL
         ELSE
            SELF.ST_PrivateOffsetVerticalDistance
             (anoffsetverticaldistance)
      END
```

Description

1) The method *ST_OffsetVerDist()* has no input parameters.
2) The null-call method *ST_OffsetVerDist()* returns the value of the *ST_PrivateOffsetVerticalDistance* attribute.
3) The method *ST_OffsetVerDist(ST_LRMeasure)* takes the following input parameters:
   a) an *ST_LRMeasure* value *anoffsetverticaldistance*.
4) For the type-preserving method *ST_OffsetVerDist(ST_LRMeasure)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return an *ST_VerOffsetExp* value with the attribute *ST_PrivateOffsetVerticalDistance* set to *anoffsetverticaldistance*.

15.12.4 ST_FeatureGeometry Methods

Purpose

Observe and mutate the attribute ST_PrivateFeatureGeometry of an ST_VerOffsetExp value.

Definition

```
CREATE METHOD ST_FeatureGeometry()
   RETURNS ST_Geometry
   FOR ST_VerOffsetExp
   RETURN SELF.ST_PrivateFeatureGeometry

CREATE METHOD ST_FeatureGeometry
   (afeaturegeometry ST_Geometry)
   RETURNS ST_VerOffsetExp
   FOR ST_VerOffsetExp
   BEGIN
      IF afeaturegeometry IS NOT NULL and
        SELF.ST_PrivateOffsetReferentDescription IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF94'
            SET MESSAGE_TEXT = 'illegal with offset referent description';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateFeatureGeometry(afeaturegeometry)
            END;
      END IF;
   END
```

Description

1) The method *ST_FeatureGeometry()* has no input parameters.

2) The null-call method *ST_FeatureGeometry()* returns the value of the *ST_PrivateFeatureGeometry* attribute.

3) The method *ST_FeatureGeometry(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value *afeaturegeometry*.

4) For the type-preserving method *ST_FeatureGeometry(ST_Geometry)*:

   Case:

   a) If *afeaturegeometry* is not the null value and *SELF.ST_PrivateOffsetReferentDescription* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with offset referent description.*

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_VerOffsetExp* value with the attribute *ST_PrivateFeatureGeometry* set to *afeaturegeometry*.

15.12.5 ST_OffsetRefDesc Methods

Purpose

Observe and mutate the attribute ST_PrivateOffsetReferentDescription of an ST_VerOffsetExp value.

Definition

```
CREATE METHOD ST_OffsetRefDesc()
   RETURNS CHARACTER VARYING(128)
   FOR ST_VerOffsetExp
   RETURN SELF.ST_PrivateOffsetReferentDescription

CREATE METHOD ST_OffsetRefDesc
   (anoffsetreferentdescription CHARACTER VARYING(128))
   RETURNS ST_VerOffsetExp
   FOR ST_VerOffsetExp
   BEGIN
      IF anoffsetreferentdescription IS NOT NULL and
        SELF.ST_PrivateFeatureGeometry IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF95'
            SET MESSAGE_TEXT = 'illegal with offset referent geometry';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateOffsetReferentDescription
                    (anoffsetreferentdescription)
            END;
      END IF;
   END
```

Description

1) The method *ST_OffsetRefDesc()* has no input parameters.

2) The null-call method *ST_OffsetRefDesc()* returns the value of the *ST_PrivateOffsetReferentDescription* attribute.

3) The method *ST_OffsetRefDesc(CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *anoffsetreferentdescription*.

4) For the type-preserving method *ST_OffsetRefDesc(CHARACTER VARYING)*:

   Case:

   a) If *anoffsetreferentdescription* is not the null value and *SELF.ST_PrivateFeatureGeometry* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – illegal with offset referent geometry.*

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return an *ST_VerOffsetExp* value with the attribute *ST_PrivateOffsetReferentDescription* set to *anoffsetreferentdescription*.

15.13 ST_VectorOffsetExp Type and Routines

15.13.1 ST_VectorOffsetExp Type

Purpose

The ST_VectorOffsetExp type specifies the vector offset for a linearly referenced location. The ST_VectorOffsetExp type is instantiable.

Definition

```
CREATE TYPE ST_VectorOffsetExp
   AS (
      ST_PrivateVectors ST_Vector ARRAY[3] DEFAULT ARRAY[]
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_VectorOffsetExp
      (avectorarray ST_Vector ARRAY[3])
      RETURNS ST_VectorOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Vectors()
      RETURNS ST_Vector ARRAY[3]
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Vectors
      (avectorarray ST_Vector ARRAY[3])
      RETURNS ST_VectorOffsetExp
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateVectors* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateVectors*.

Description

1) The *ST_VectorOffsetExp* type provides for public use:
   a) a method *ST_VectorOffsetExp(ST_Vector ARRAY)*,
   b) a method *ST_Vectors()*,
   c) a method *ST_Vectors(ST_Vector ARRAY).*
2) The *ST_PrivateVectors* attribute contains the *ST_Vector* ARRAY collection of up to three offset vector values.
3) An offset vector value specifies the distance and bearing of the offset from the linear element being measured to the position being specified. A 0 (zero) length vector is to be interpreted as not having a vector displacement from the linear element being measured.

4) Up to three offset vector values are permitted to enable the offset direction to be defined in terms of up to three component base offset vectors.

5) Unlike *ST_LatOffsetExp* and *ST_VerOffsetExp*, *ST_VectorOffsetExp* does not allow an offset referent.

15.13.2 ST_VectorOffsetExp Methods

Purpose

Return an ST_VectorOffsetExp value constructed from the specified ST_Vector ARRAY collection of up to three offset vectors.

Definition

```
CREATE CONSTRUCTOR METHOD ST_VectorOffsetExp
   (avectorarray ST_Vector ARRAY[3])
   RETURNS ST_VectorOffsetExp
   FOR ST_VectorOffsetExp
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_VectorOffsetExp(ST_Vector ARRAY)* takes the following input parameters:
   a) an *ST_Vector* ARRAY value *avectorarray.*
2) The null-call type-preserving SQL-invoked constructor method *ST_VectorOffsetExp(ST_Vector ARRAY)* returns an *ST_VectorOffsetExp* value with:
   a) The *ST_PrivateVectors* attribute set to *avectorarray.*

15.13.3 ST_Vectors Methods

Purpose

Observe and mutate the attribute ST_PrivateVectors of an ST_VectorOffsetExp value.

Definition

```
CREATE METHOD ST_Vectors()
   RETURNS ST_Vector ARRAY[3]
   FOR ST_VectorOffsetExp
   RETURN SELF.ST_PrivateVectors

CREATE METHOD ST_Vectors
   (avectorarray ST_Vector ARRAY[3])
   RETURNS ST_VectorOffsetExp
   FOR ST_VectorOffsetExp
   BEGIN
      IF avectorarray IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateVectors(avectorarray)
            END;
      END IF;
   END
```

Description

1) The method *ST_Vectors()* has no input parameters.
2) The null-call method *ST_Vectors()* returns the value of the *ST_PrivateVectors* attribute.
3) The method *ST_Vectors(ST_Vector ARRAY)* takes the following input parameters:
   a) an *ST_Vector* ARRAY value *avectorarray*.
4) For the type-preserving method *ST_Vectors(ST_Vector ARRAY)*:

   Case:

   a) If *avectorarray* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return an *ST_VectorOffsetExp* value with the attribute *ST_PrivateVectors* set to *avectorarray*.

15.14 Linear Referencing Well-Known Text

15.14.1 <position expression text representation>

Purpose

This subclause contains the definition of <position expression text representation>.

Description

1) The well-known text representation of an *ST_PositionExp* value is defined by the following BNF for <position expression text representation>.

```
<position expression text representation> ::=
     POSEXP
        <left paren> <linear element text representation> <right paren>
        <comma>
        <left paren> <lrm text representation> <right paren>
        <comma>
        <left paren> <distance expression text representation>
           <right paren>
```

a) <position expression text representation> is the well-known text representation for an *ST_PositionExp* value. Let *DE* be the *ST_DistanceExp* distance expression value produced by the immediately contained <distance expression text representation>. Case:

i) If <linear element text body> immediately contains a <linear element id representation> and <lrm text representation> immediately contains an <lrm text> which immediately contains an <lrm id representation>, then

1) let *LEID* be the INTEGER linear element ID value produced by the <linear element id representation>.

2) let *LRMID* be the INTEGER LRM ID value produced by the <lrm id representation>.

3) <position expression text representation> produces an *ST_PositionExp* value as the result of the value expression: *NEW ST_PositionExp(LEID, LRMID, DE)*.

ii) If <linear element text body> immediately contains a <linear element id representation> and <lrm text representation> immediately contains an <lrm text> which immediately contains an <lrm representation>, then

1) let *LEID* be the INTEGER linear element ID value produced by the <linear element id representation>.

2) let *LRM* be the *ST_LRM* LRM value produced by the <lrm representation>.

3) <position expression text representation> produces an *ST_PositionExp* value as the result of the value expression: *NEW ST_PositionExp(LEID, LRM, DE)*.

iii) If <linear element text body> immediately contains a <linear element representation> and <lrm text representation> immediately contains an <lrm text> which immediately contains an <lrm id representation>, then:

1) let *LE* be the *ST_LinearElement* linear element value produced by the <linear element representation>.

2) let *LRMID* be the INTEGER LRM ID value produced by the <lrm id representation>.

3) <position expression text representation> produces an *ST_PositionExp* value as the result of the value expression: *NEW ST_PositionExp(LE, LRMID, DE)*.

iv) Otherwise,

1) let *LE* be the *ST_LinearElement* linear element value produced by the <linear element representation>.

2) let *LRM* be the *ST_LRM* LRM value produced by the <lrm representation>.

3) <position expression text representation> produces an *ST_PositionExp* value as the result of the value expression: *NEW ST_PositionExp(LE, LRM, DE)*.

15.14.2 <linear element text representation>

Purpose

This subclause contains the definition of <linear element text representation>.

Description

1) The well-known text representation of an *ST_LinearElement* value and those of its subtypes *ST_LRFeature, ST_LRCurve,* and *ST_LRDirectedEdge* are defined by the following BNF for <linear element text representation>. The well-known text representations for *ST_LRMeasure, ST_StartValue,* and *ST_Referent* values used by *ST_LinearElement* and its subtypes are also included.

```
<linear element text representation> ::=
     LINEARELEMENT <linear element text body>

<linear element text body> ::=
     <linear element id representation>
   | <linear element representation>

<linear element id representation> ::=
     LEID <leid representation>

<leid representation> ::=
     <positive integer>

<linear element representation> ::=
     <lr feature text representation>
   | <lr curve text representation>
   | <lr directed edge text representation>

<lr feature text representation> ::=
     LRFEATURE <linear element text> <comma> <lr feature text>

<lr curve text representation> ::=
     LRCURVE <linear element text> <comma> <lr curve text>

<lr directed edge text representation> ::=
     LRDIRECTEDEDGE <linear element text> <comma> <lr directed edge text>

<linear element text> ::=
     <linear element id representation>
        <comma> <default lrm>
        <comma> <default measure>
        <comma> <linear element type>
        [<comma> <start values>]

<default lrm> ::=
     DEFAULT <lrm id representation>

<default measure> ::=
     DEFAULT <measure representation>

<linear element type> ::=
     LETYPE <text>

<start values> ::=
     STARTVALUES <left paren> <start value>
        { <comma> <start value> }... <right paren>

<start value> ::=
     <left paren> <lrm id representation> <comma>
        <measure text> <right paren>

<measure representation> ::=
     MEASURE <measure text>

<measure text> ::=
```



```
     <measure value> [<unit of measure>]

<measure value> ::=
     <number>

<unit of measure> ::=
     <letters>

<lr feature text> ::=
     <feature id representation> [<referents>]

<feature id representation> ::=
     FID <fid representation>

<fid representation> ::=
     <text>

<referents> ::=
     REFERENTS <left paren> <referent>
        { <comma> <referent> }... <right paren>

<referent> ::=
     <left paren> <referent text representation> <right paren>

<referent text representation> ::=
     <referent name text>
        <comma> <referent type>
        [<comma> <referent position>]
        [<comma> <referent location>]

<referent name text> ::=
     NAME <referent name>

<referent name> ::=
     <text>

<referent type> ::=
     TYPE <text>

<referent position> ::=
     POSITION <point text representation>

<referent location> ::=
     LOCATION <position expression text representation>

<lr curve text> ::=
     <curve text representation>

<lr directed edge text> ::=
     <topology type> <topology or network name> <edge or link id>

<topology type> ::=
     <e l>

<e l> ::=
     E
   | L

<topology or network name> ::=
     <text>

<edge or link id> ::=
     <unsigned integer>
```

a) Case:

i) If <linear element text body> immediately contains a <linear element id representation>, then <linear element text body> produces an INTEGER value equal to the linear element ID attribute value of some *ST_LinearElement* value.

ii) Otherwise, <linear element text body> produces an *ST_LinearElement* value specified by the immediately contained <linear element representation>.

b) <leid representation> is the well-known text representation for an INTEGER value for referencing an *ST_LinearElement*. <leid representation> produces an INTEGER linear element ID value specified by the immediately contained <positive integer>.

c) <linear element representation> is the well-known text representation for an instantiable subtype of *ST_LinearElement*. Let *LEID* be the INTEGER linear element ID value produced by the <linear element id representation>, *LRMID* the INTEGER default LRM ID value produced by <default lrm>, *M* the ST_LRMeasure default measure value produced by <default measure>, *LET* the CHARACTER VARYING linear element type value produced by <linear element type> and *SVA* the *ST_StartValue* ARRAY start value collection value produced by <start values>, all in the same <linear element text>. If <start values> is not immediately contained in this <linear element text>, then let *SVA* be NULL. Case:

i) If <linear element representation> immediately contains an <lr feature text representation>, then <linear element representation> produces an *ST_LRFeature* specified by the immediately contained <lr feature text representation>.

1) let *FID* be the CHARACTER VARYING feature ID value produced by <lr feature text>.

2) let *RA* be the ST_Referent ARRAY referent collection value produced by the <lr feature text>. If <referents> is not immediately contained in <lr feature text>, then let *RA* be NULL.

3) <linear element representation> produces an *ST_LRFeature* value as the result of the value expression: *NEW ST_LRFeature(LEID, LRMID, M, LET, SVA, FID, RA)*.

ii) If <linear element representation> immediately contains an <lr curve text representation>, then <linear element representation> produces an *ST_LRCurve* specified by the immediately contained <lr curve text representation>.

1) let *C* be the *ST_Curve* curve value produced by <lr curve text>.

2) <linear element representation> produces an *ST_LRCurve* value as the result of the value expression: *NEW ST_LRCurve(LEID, LRMID, M, LET, SVA, C)*.

iii) Otherwise, <linear element representation> produces an *ST_LRDirectedEdge* value specified by the immediately contained <lr directed edge text representation>.

1) let *TT* be the CHARACTER topology type value produced by <lr directed edge text>.

2) let *TNN* be the CHARACTER VARYING topology or network name value produced by <lr directed edge text>.

3) let *ELID* be the INTEGER edge or link ID value produced by the <lr directed edge text>.

4) <linear element representation> produces an *ST_LRDirectedEdge* value as the result of the value expression: *NEW ST_LRDirectedEdge(LEID, LRMID, M, LET, SVA, TT, TNN, ELID)*.

d) <lr feature text representation> is the well-known text representation for an *ST_LRFeature* value. <lr feature text representation> produces an *ST_LRFeature* value specified by the immediately contained <linear element text> and <lr feature text>.

e) <lr curve text representation> is the well-known text representation for an *ST_LRCurve* value. <lr curve text representation> produces an *ST_LRCurve* value specified by the immediately contained <linear element text> and <lr curve text>.

f) <lr directed edge text representation> is the well-known text representation for an *ST_LRDirectedEdge* value. <lr directed edge text representation> produces an *ST_LRDirectedEdge* value specified by the immediately contained <linear element text> and <lr directed edge text>.

g) <linear element text> produces the *ST_LinearElement* attributes inherited by its *ST_LRFeature, ST_LRCurve* and *ST_LRDirectedEdge* subtypes from the immediately contained <linear element id representation>, <default lrm>, <default measure>, <linear element type> and optional <start values>.

h) <default lrm> is the well-known text representation for an *ST_LinearElement* default LRM attribute value. <default lrm> produces an *ST_LinearElement* default LRM attribute value from the immediately contained <lrm id representation>.

i) <default measure> is the well-known text representation for an *ST_LinearElement* default measure attribute value. <default measure> produces an *ST_LinearElement* default measure attribute value from the immediately contained <measure representation>.

j) <linear element type> is the well-known text representation for an *ST_LinearElement* linear element type attribute value. <linear element type> produces an *ST_LinearElement* linear element type attribute value from the immediately contained <text>.

k) <start values> is the well-known text representation for an *ST_LinearElement* start values attribute value. <start values> produces an *ST_LinearElement* start values attribute value as an *ST_StartValue* ARRAY from the immediately contained <start value>s.

l) <lr feature text> is the well-known text representation for the subtype-specific part of an *ST_LRFeature* value. <lr feature text> produces the subtype-specific part of an *ST_LRFeature* value specified by the immediately contained <feature id representation> and optional <referents>.

m) <fid representation> is the well-known text representation for a CHARACTER VARYING value for referencing an *ST_LRFeature*. <fid representation> produces a CHARACTER VARYING feature ID value specified by the immediately contained <text>.

n) <referents> is the well-known text representation for an *ST_LRFeature* referents attribute value. <referents> produces an *ST_LRFeature* referents attribute value as an *ST_Referent* ARRAY from the immediately contained <referent>s.

o) <referent text representation> is the well-known text representation for an *ST_Referent* value.

   i) Let *RN* the CHARACTER VARYING referent name value produced by <referent name text>, *RT* the CHARACTER VARYING referent type value produced by <referent type>, *RP* the *ST_Point* referent position value produced by <referent position> and *RL* the *ST_PositionExp* referent location value produced by <referent location>, all in the same <referent text representation>.

   ii) If <referent position> is not immediately contained in this <referent text representation>, then let *RP* be NULL.

   iii) If <referent location> is not immediately contained in this <referent text representation>, then let *RL* be NULL.

   iv) <referent text representation> produces an *ST_Referent* value as the result of the value expression: *NEW ST_Referent(RN, RT, RP, RL)*.

p) <referent name> is the well-known text representation for an *ST_Referent* referent name attribute value. <referent name> produces an *ST_Referent* referent name attribute value from the immediately contained <text>.

q) <referent type> is the well-known text representation for an *ST_Referent* referent type attribute value. <referent type> produces an *ST_Referent* referent type attribute value from the immediately contained <text>.

r) <referent position> is the well-known text representation for an *ST_Referent* referent position attribute value. <referent position> produces an *ST_Referent* referent position attribute value from the immediately contained <point text representation>.

s) <referent location> is the well-known text representation for an *ST_Referent* referent location attribute value. <referent location> produces an *ST_Referent* referent location attribute value from the immediately contained <position expression text representation>.

t) <lr curve text> is the well-known text representation for the subtype-specific part of an *ST_LRCurve* value. <lr curve text> produces the subtype-specific part of an *ST_LRCurve* value specified by the immediately contained <curve text representation>.

u) <lr directed edge text> is the well-known text representation for the subtype-specific part of an *ST_LRDirectedEdge* value. <lr directed edge text> produces the subtype-specific part of an *ST_LRDirectedEdge* value specified by the immediately contained <topology type>, <topology or network name> and <edge or link id>.

v) <topology type> is the well-known text representation for an *ST_LRDirectedEdge* topology type attribute value. <topology type> produces an *ST_LRDirectedEdge* topology type attribute value from the immediately contained <e l>.

w) <topology or network name> is the well-known text representation for an *ST_LRDirectedEdge* topology or network name attribute value. <topology or network name> produces an *ST_LRDirectedEdge* topology or network name attribute value from the immediately contained <text>.

x) <edge or link id> is the well-known text representation for an *ST_LRDirectedEdge* edge or link id attribute value. <edge or link id> produces an *ST_LRDirectedEdge* edge or link id attribute value from the immediately contained <unsigned integer>.

15.14.3 <lrm text representation>

Purpose

This subclause contains the definition of <lrm text representation>.

Description

1) The well-known text representation of an *ST_LRM* value is defined by the following BNF for <lrm text representation>.

```
<lrm text representation> ::=
     LRM <lrm text body>

<lrm text body> ::=
     <lrm id representation>
   | <lrm representation>

<lrm id representation> ::=
     LRMID <lrmid representation>

<lrmid representation> ::=
     <positive integer>

<lrm representation> ::=
     <lrm id representation>
        <comma> <lrm name>
        <comma> <lrm type>
        <comma> <lrm unit of measure>
        [<comma> <lrm constraints>]
        [<comma> <lrm offset attributes>]

<lrm offset attributes> ::=
     <offset unit of measure>
        [<comma> <positive lateral offset direction>]
        [<comma> <positive vertical offset direction>]

<lrm name> ::=
     LRMNAME <text>

<lrm type> ::=
     LRMTYPE <text>

<lrm unit of measure> ::=
     LRMUOM <text>

<lrm constraints> ::=
     LRMCONSTRAINTS <left paren> <constraint>
        { <comma> <constraint> }... <right paren>

<constraint> ::=
     <text>

<offset unit of measure> ::=
     LRMOFFSETUOM <text>

<positive lateral offset direction> ::=
     POSLATOFFDIR <text>

<positive vertical offset direction> ::=
     POSVEROFFDIR <text>
```

a) Let *LRMID* be the INTEGER LRM ID value produced by the <lrm id representation> and let *M* be the *ST_LRMeasure* lr measure value produced by the <measure text>, all in the same <start value>. <start value> produces an *ST_StartValue* value as the result of the value expression: *NEW ST_StartValue(LRMID,M)*.

b) For <measure value> and <measure units> in the same <measure text>, let *M* be the DOUBLE PRECISION measure value produced by <measure value>. If <measure text> immediately contains a <unit of measure> then let *U* be the CHARACTER VARYING unit of measure value produced by <unit of measure>, otherwise let *U* be NULL. <measure text> produces an *ST_LRMeasure* value as the result of the value expression: *NEW ST_LRMeasure(M,U)*.

c) Case:

   i) If <lrm text body> immediately contains a <lrm id representation>, then <lrm text body> produces an INTEGER value equal to the LRM ID attribute value of some *ST_LRM* value.

   ii) Otherwise, <lrm text body> produces an *ST_LRM* value specified by the immediately contained <lrm representation>.

d) <lrm id representation> is the well-known text representation for an INTEGER value for referencing an *ST_LRM*. <lrm id representation> produces an INTEGER LRM ID value specified by the immediately contained <positive integer>.

e) <lrm representation> is the well-known text representation for an *ST_LRM* value.

   i) Let *LRMID* be the INTEGER LRM ID value produced by the <lrm id representation>, *LN* the CHARACTER VARYING LRM name value produced by <lrm name>, *LT* the CHARACTER VARYING LRM type value produced by <lrm type>, *LU* the CHARACTER VARYING LRM unit of measure value produced by <lrm unit of measure>, *LCA* the CHARACTER VARYING ARRAY constraint collection value produced by <lrm constraints>, *LOU* the CHARACTER VARYING LRM offset unit of measure value produced by <offset unit of measure>, *PLOD* the CHARACTER VARYING LRM positive lateral offset direction value produced by <positive lateral offset direction> and *PVOD* the CHARACTER VARYING LRM positive vertical offset direction value produced by <positive vertical offset direction>, all in the same <lrm representation>.

   ii) If <lrm constraints> is not immediately contained in this <lrm representation>, then let *LCA* be NULL.

   iii) If <lrm offset attributes> is not immediately contained in this <lrm representation>, then let *LOU, PLOD* and *PVOD* be NULL.

   iv) If <positive lateral offset direction> is not immediately contained in an <lrm offset attributes> immediately contained in the <lrm representation>, then let *PLOD* be NULL.

   v) If <positive vertical offset direction> is not immediately contained in an <lrm offset attributes> immediately contained in the <lrm representation>, then let *PVOD* be NULL.

   vi) <lrm representation> produces an *ST_LRM* value as the result of the value expression: *NEW ST_LRM(LRMID, LN, LT, LU, LCA, LOU, PLOD, PVOD)*.

f) <lrm name> is the well-known text representation for an *ST_LRM* LRM name attribute value. <lrm name> produces an *ST_LRM* LRM name attribute value from the immediately contained <text>.

g) <lrm type> is the well-known text representation for an *ST_LRM* LRM type attribute value. <lrm type> produces an *ST_LRM* LRM type attribute value from the immediately contained <text>.

h) <lrm unit of measure> is the well-known text representation for an *ST_LRM* LRM unit of measure attribute value. <lrm unit of measure> produces an *ST_LRM* LRM unit of measure attribute value from the immediately contained <text>.

i) <lrm constraints> is the well-known text representation for an *ST_LRM* LRM constraints attribute value. <lrm constraints> produces an *ST_LRM* LRM constraints attribute value from the collection of immediately contained <constraint>s.

j) <constraint> is the well-known text representation for an *ST_LRM* LRM constraint value. <constraint> produces an *ST_LRM* LRM constraint value from the immediately contained <text>.

k) <offset unit of measure> is the well-known text representation for an *ST_LRM* LRM offset unit of measure attribute value. <offset unit of measure> produces an *ST_LRM* LRM offset unit of measure attribute value from the immediately contained <text>.

l) <positive lateral offset direction> is the well-known text representation for an *ST_LRM* LRM positive lateral offset direction attribute value. <positive lateral offset direction> produces an *ST_LRM* LRM positive lateral offset direction attribute value from the immediately contained <text>.

m) <positive vertical offset direction> is the well-known text representation for an *ST_LRM* LRM positive vertical offset direction attribute value. <positive vertical offset direction> produces an *ST_LRM* LRM positive vertical offset directionattribute value from the immediately contained <text>.

15.14.4 <distance expression text representation>

Purpose

This subclause contains the definition of <distance expression text representation>.

Description

1) The well-known text representation of an *ST_DistanceExpression* value is defined by the following BNF for <distance expression text representation>. The well-known text representations for *ST_LatOffsetExp, ST_VerOffsetExp,* and *ST_VectorOffsetExp* values used by *ST_DistanceExpression* are also included.

```
<distance expression text representation> ::=
     DISEXP <distance expression representation>

<distance expression representation> ::=
     <distance along>
        [<from referent>]
        [<towards referent>]
        [<comma> <offset expression text representation>]

<distance along> ::=
     <measure representation>

<from referent> ::=
     FROM [<from referent feature id>] <from referent name>

<from referent feature id> ::=
     <fid representation>

<from referent name> ::=
     <referent name>

<towards referent> ::=
     TOWARDS [<towards referent feature id>] <towards referent name>

<towards referent feature id> ::=
     <fid representation>

<towards referent name> ::=
     <referent name>

<offset expression text representation> ::=
     <lateral offset expression text representation>
   | <vertical offset expression text representation>
   | <lateral offset expression text representation> <comma>
        <vertical offset expression text representation>
   | <vector offset expression text representation>

<lateral offset expression text representation> ::=
     LATERALOFFSET <lateral offset expression text>

<lateral offset expression text> ::=
     <offset lateral distance> [<lateral offset referent text>]

<offset lateral distance> ::=
     <measure text>

<lateral offset referent text> ::=
     FROM <lateral offset referent text body>

<lateral offset referent text body> ::=
     <feature geometry>
   | <offset referent description>

<feature geometry> ::=
     <well-known text representation>

<offset referent description> ::=
```

```
     <text>

<vertical offset expression text representation> ::=
     VERTICALOFFSET <vertical offset expression text>

<vertical offset expression text> ::=
     <offset vertical distance> [<vertical offset referent text>]

<offset vertical distance> ::=
     <measure text>

<vertical offset referent text> ::=
     FROM <vertical offset referent text body>

<vertical offset referent text body> ::=
     <feature geometry>
   | <offset referent description>

<vector offset expression text representation> ::=
     VECTOROFFSETS <vector offset expression text>

<vector offset expression text> ::=
     <vectors>

<vectors> ::=
     <left paren> <vector text representation>
        [<comma> <vector text representation>]
        [<comma> <vector text representation>]
        <right paren>

<point text representation> ::=
     !! See Subclause 5.1.67, "<well-known text representation>"

<curve text representation> ::=
     !! See Subclause 5.1.67, "<well-known text representation>"

<well-known text representation> ::=
     !! See Subclause 5.1.67, "<well-known text representation>"

<vector text representation> ::=
     !! See Subclause 16.2.22, "<well-known text representation>"
```

a) <distance expression representation> is the well-known text representation for ST_DistanceExp value.

   i) Let *DA* be the *ST_LRMeasure* distance along value produced by <distance along>, *FRFID* the CHARACTER VARYING from referent feature id and *FRN* the CHARACTER VARYING from referent name values produced by <from referent>, *TRFID* the CHARACTER VARYING towards referent feauture id and *TRN* the CHARACTER VARYINGtowards referent name values produced by <towards referent> and *LOE* the *ST_LatOffsetExp* lateral offset expression, *VOE* the *ST_VerOffsetExp* vertical offset expression and *VcOE* the *ST_VectorOffsetExp* vector offset expression values produced by <offset expression text representation>, all in the same <distance expression representation>.

   ii) If <from referent> is not immediately contained in this <distance expression representation>, then let *FRFID*, *FRID, TRFID* and *TRID* be NULL.

   iii) If <towards referent> is not immediately contained in this <distance expression representation>, then let *TRFID* and *TRID* be NULL.

   iv) If <from referent feature id> is not immediately contained in a <from referent> immediately contained in this <distance expression representation>, then let *FRFID* be NULL.

   v) If <towards referent feature id> is not immediately contained in a <towards referent> immediately contained in this <distance expression representation>, then let *TRFID* be NULL.

   vi) If <offset expression text representation> is not immediately contained in this <distance expression representation>, then let *LOE*, *VOE* and *VcOE* be NULL.

vii) If <lateral offset expression text representation> is not immediately contained in the <offset expression text representation> immediately contained in the <distance expression representation>, then let *LOE* be NULL.

viii) If <vertical offset expression text representation> is not immediately contained in the <offset expression text representation> immediately contained in the <distance expression representation>, then let *VOE* be NULL.

ix) If <vector offset expression text representation> is not immediately contained in the <offset expression text representation> immediately contained in the <distance expression representation>, then let *VcOE* be NULL.

x) Case:

1) if *VcOE* is NULL, then <distance expression representation> produces an *ST_DistanceExp* value as the result of the value expression: *NEW ST_DistanceExp (DA, FRFID*, *FRID, TRFID*, *TRID, LOE, VOE)*.

2) otherwise, <distance expression representation> produces an *ST_DistanceExp* value as the result of the value expression: *NEW ST_DistanceExp (DA, FRFID*, *FRID, TRFID*, *TRID, VcOE)*.

b) <distance along> is the well-known text representation for an *ST_DistanceExp* distance along attribute value. <referent name> produces an *ST_DistanceExp* distance along attribute value from the immediately contained <measure representation>.

c) <from referent feature id> is the well-known text representation for an *ST_DistanceExp* from referent feature id attribute value. <from referent feature id> produces an *ST_DistanceExp* from referent feature id attribute value from the immediately contained <fid representation>.

d) <from referent name> is the well-known text representation for an *ST_DistanceExp* from referent name attribute value. <from referent name> produces an *ST_DistanceExp* from referent nameattribute value from the immediately contained <referent name>.

e) <towards referent feature id> is the well-known text representation for an *ST_DistanceExp* towards referent feature id attribute value. <towards referent feature id> produces an *ST_DistanceExp* towards referent feature id attribute value from the immediately contained <fid representation>.

f) <towards referent name> is the well-known text representation for an *ST_DistanceExp* towards referent nameattribute value. <towards referent name> produces an *ST_DistanceExp* towards referent nameattribute value from the immediately contained <referent name>.

g) <offset expression text representation> is the well-known text representation for the offset expressions for an *ST_DistanceExp* value. Case:

i) If only <lateral offset expression text representation> is immediately contained in <offset expression text representation>, then the *ST_DistanceExp* value will only contain a lateral offset.

ii) If only <vertical offset expression text representation> is immediately contained in <offset expression text representation>, then the *ST_DistanceExp* value will only contain a vertical offset.

iii) If both <lateral offset expression text representation> and <vertical offset expression text representation> are immediately contained in <offset expression text representation>, then the *ST_DistanceExp* value will contain both a lateral and a vertical offset.

iv) Otherwise, the *ST_DistanceExp* value will only contain a vector offset.

h) <lateral offset expression text> is the well-known text representation for an *ST_LatOffsetExp* value.

i) Let *OLD* be the *ST_LRMeasure* offset lateral distance value produced by <measure text> and let *FG* be the *ST_Geometry* feature geometry and *ORD* the CHARACTER VARYING offset referent description values produced by <lateral offset referent text>, all in the same <lateral offset expression text>.

ii) If <lateral offset referent text> is not immediately contained in this <lateral offset expression text>, then let *FG* and *ORD* be NULL.

iii) If <feature geometry> is not immediately contained in the <lateral offset referent text body> immediately contained in the <lateral offset referent text> immediately contained in the <lateral offset expression text>, then let *FG* be NULL.

iv) If <offset referent description> is not immediately contained in the <lateral offset referent text body> immediately contained in the <lateral offset referent text> immediately contained in the <lateral offset expression text>, then let *ORD* be NULL.

v) Case:

1) if *FG* and *ORD* are both NULL, then <lateral offset expression text> produces an *ST_LatOffsetExp* value as the result of the value expression: *NEW ST_LatOffsetExp(OLD)*.

2) if *FG* is NULL, then <lateral offset expression text> produces an *ST_LatOffsetExp* value as the result of the value expression: *NEW ST_LatOffsetExp(OLD, ORD)*.

3) otherwise, <lateral offset expression text> produces an *ST_LatOffsetExp* value as the result of the value expression: *NEW ST_LatOffsetExp(OLD, FG)*.

i) <lateral offset referent text body> is the well-known text representation for the lateral offset referent for an *ST_DistanceExp* value. Case:

i) If <feature geometry> is immediately contained in <lateral offset referent text body>, then the lateral offset referent of the *ST_DistanceExp* value will be a feature geometry.

ii) Otherwise, the lateral offset referent of the *ST_DistanceExp* value will be an offset referent description.

j) <feature geometry> is the well-known text representation for an *ST_LatOffsetExp* feature geometry attribute value. <referent name> produces an *ST_LatOffsetExp* feature geometry attribute value from the immediately contained <well-known text representation>.

k) <offset referent description> is the well-known text representation for an *ST_LatOffsetExp* offset referent description attribute value. <offset referent description> produces an *ST_LatOffsetExp* offset referent description attribute value from the immediately contained <text>.

l) <vertical offset expression text> is the well-known text representation for an *ST_VerOffsetExp* value.

i) Let *OVD* be the *ST_LRMeasure* offset vertical distance value produced by <measure text> and let *FG* be the *ST_Geometry* feature geometry and *ORD* the CHARACTER VARYING offset referent description values produced by <vertical offset referent text>, all in the same <vertical offset expression text>.

ii) If <vertical offset referent text> is not immediately contained in this <vertical offset expression text>, then let *FG* and *ORD* be NULL.

iii) If <feature geometry> is not immediately contained in the <vertical offset referent text body> immediately contained in the <vertical offset referent text> immediately contained in the <vertical offset expression text>, then let *FG* be NULL.

iv) If <offset referent description> is not immediately contained in the <vertical offset referent text body> immediately contained in the <vertical offset referent text> immediately contained in the <vertical offset expression text>, then let *ORD* be NULL.

v) Case:

1) if *FG* and *ORD* are both NULL, then <vertical offset expression text> produces an *ST_VerOffsetExp* value as the result of the value expression: *NEW ST_VerOffsetExp(OVD)*.

2) if *FG* is NULL, then <vertical offset expression text> produces an *ST_VerOffsetExp* value as the result of the value expression: *NEW ST_VerOffsetExp(OVD, ORD)*.

3) otherwise, <vertical offset expression text> produces an *ST_VerOffsetExp* value as the result of the value expression: *NEW ST_VerOffsetExp(OVD, FG)*.

m) <vertical offset referent text body> is the well-known text representation for the vertical offset referent for an *ST_DistanceExp* value. Case:

   i) If <feature geometry> is immediately contained in <vertical offset referent text body>, then the vertical offset referent of the *ST_DistanceExp* value will be a feature geometry.

   ii) Otherwise, the vertical offset referent of the *ST_DistanceExp* value will be an offset referent description.

n) <vector offset expression text> is the well-known text representation for an *ST_VectorOffsetExp* value.

   i) Let *VA* be the *ST_Vector* ARRAY of offset vector values constructed from the *ST_Vector* values produced by the <vector text representation>s, all in the same <vectors> immediately contained in <vector offset expression text>.

   ii) <vector offset expression text> produces an *ST_VectorOffsetExp* value as the result of the value expression: *NEW ST_VectorOffsetExp (VA)*.

15.14.5 Additional BNF Productions

Purpose

This subclause contains the definition of additional BNF productions used in linear referencing.

Description

1) Additional well-known text representations used in linear referencing are defined by the following BNF.

```
<number> ::=
      <exact numeric literal>
   |  <approximate numeric literal>

<exact numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<approximate numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<text> ::=
     <double quote> <letters> <double quote>

<double quote> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<letters> ::= <letter>...

<letter> ::=
     <simple Latin letter>
   | <digit>
   | <special>


<simple Latin letter> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<digit> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<special> ::=
     <left paren>
   | <right paren>
   | <minus sign>
   | <underscore>
   | <period>
   | <quote>
   | <space>

<left paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<minus sign> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<underscore> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075
```



```
<period> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<quote> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<space> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<comma> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<unsigned integer> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<positive integer> ::=
     <non-zero digit> {<digit>}...

<non-zero digit> ::=
     1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<digit> ::=
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

a) The list of keywords for linear referencing is:

DEFAULT
DISEXP

E

FID
FROM

L
LATERALOFFSET
LEID
LETYPE
LINEARELEMENT
LOCATION
LRCURVE
LRDIRECTEDEDGE
LRFEATURE
LRM
LRMCONSTRAINTS
LRMID
LRMNAME
LRMOFFSETUOM
LRMTYPE
LRMUOM

MEASURE

NAME

POSEXP
POSITION
POSLATOFFDIR
POSVEROFFDIR

REFERENTS

STARTVALUES

TOWARDS
TYPE

VECTOROFFSETS
VERTICALOFFSET

16 Angle and Direction Types

16.1 ST_Angle Type and Routines

16.1.1 ST_Angle Type

Purpose

The ST_Angle type is used to express circular measurement or to measure the degree of separation of two intersecting lines.

Definition

```
CREATE TYPE ST_Angle
   AS (
      ST_PrivateRadians DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Angle
      (angle DOUBLE PRECISION)
      RETURNS ST_Angle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Angle
      (units CHARACTER(1),
       angle DOUBLE PRECISION)
      RETURNS ST_Angle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Angle
      (degrees INTEGER, minutes DOUBLE PRECISION)
      RETURNS ST_Angle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Angle
      (degrees INTEGER, minutes INTEGER, seconds DOUBLE PRECISION)
      RETURNS ST_Angle
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Angle
   (atpoint ST_Point, apoint ST_Point, anotherpoint ST_Point)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Angle
   (adirection ST_Direction,
    anotherdirection ST_Direction)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Angle
   (aline ST_LineString,
    anotherline ST_LineString)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Angle
   (awkt CHARACTER VARYING(ST_MaxAngleAsText))
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Angle
   (agml CHARACTER LARGE OBJECT(ST_MaxAngleAsGML))
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Radians()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Radians
   (radians DOUBLE PRECISION)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Degrees()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Degrees
   (degrees DOUBLE PRECISION)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_DegreeComponent()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_MinuteComponent()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SecondComponent()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_String()
   RETURNS CHARACTER VARYING(ST_MaxAngleString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_String
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxAngleString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_String
   (dms CHARACTER VARYING(ST_MaxAngleString))
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Gradians()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Gradians
   (gradians DOUBLE PRECISION)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Add
   (anangle ST_Angle)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Subtract
   (anangle ST_Angle)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Multiply
   (afactor DOUBLE PRECISION)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Divide
   (adivisor DOUBLE PRECISION)
   RETURNS ST_Angle
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_AsText()
   RETURNS CHARACTER VARYING(ST_MaxAngleAsText)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxAngleAsGML))
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxAngleAsGML)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxAngleString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of the *ST_PrivateRadians* attribute of an *ST_Angle* value.
2) *ST_MaxAngleAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Angle* value.
3) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.
4) The attribute *ST_PrivateRadians* is not for public use. There are no GRANT statements granting EXECUTE privilege to the observer or mutator method for *ST_PrivateRadians*.

Description

1) The *ST_Angle* type provides for public use:
   a) a method *ST_Angle(DOUBLE PRECISION)*,
   b) a method *ST_Angle(CHARACTER, DOUBLE PRECISION)*,
   c) a method *ST_Angle(INTEGER, DOUBLE PRECISION)*,
   d) a method *ST_Angle(INTEGER, INTEGER, DOUBLE PRECISION)*,
   e) a method *ST_Angle(ST_Point, ST_Point, ST_Point)*,
   f) a method *ST_Angle(ST_Direction, ST_Direction)*,
   g) a method *ST_Angle(ST_LineString, ST_LineString)*,
   h) a method *ST_Angle(CHARACTER VARYING)*,
   i) a method *ST_Angle(CHARACTER LARGE OBJECT)*,
   j) a method *ST_Radians()*,
   k) a method *ST_Radians(DOUBLE PRECISION)*,
   l) a method *ST_Degrees()*,
   m) a method *ST_Degrees(DOUBLE PRECISION)*,
   n) a method *ST_DegreeComponent()*,
   o) a method *ST_MinuteComponent()*,

p) a method *ST_SecondComponent()*,

q) a method *ST_String()*,

r) a method *ST_String(INTEGER)*,

s) a method *ST_String(CHARACTER VARYING)*,

t) a method *ST_Gradians()*,

u) a method *ST_Gradians(DOUBLE PRECISION)*,

v) a method *ST_Add(ST_Angle),*

w) a method *ST_Subtract(ST_Angle)*,

x) a method *ST_Multiply(DOUBLE PRECISION)*,

y) a method *ST_Divide(DOUBLE PRECISION)*,

z) a method *ST_AsText()*,

aa) a method *ST_GMLToSQL(CHARACTER LARGE OBJECT)*,

ab) a method *ST_AsGML()*,

ac) an ordering function *ST_OrderingCompare(ST_Angle, ST_Angle)*,

ad) an SQL Transform group *ST_WellKnownText*,

ae) an SQL Transform group *ST_WellKnownBinary*,

af) an SQL Transform group *ST_GML*,

ag) a function *ST_AngleFromText(CHARACTER VARYING)*,

ah) a function *ST_AngleFromGML(CHARACTER LARGE OBJECT)*.

2) The attribute *ST_PrivateRadians* contains the measurement of the angle expressed in radians.

3) The direction of an angle is not specified.

16.1.2 ST_Angle Methods

Purpose

Return a specified ST_Angle value.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Angle
   (angle DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   RETURN NEW ST_Angle().ST_PrivateRadians(angle)

CREATE CONSTRUCTOR METHOD ST_Angle
   (units CHARACTER(1),
    angle DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF (units <> 'R') AND
         (units <> 'D') AND
         (units <> 'G') THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument: invalid angle units';
      ELSE
         RETURN NEW ST_Angle().ST_PrivateRadians(
            CASE
               WHEN units = 'R' THEN angle
               WHEN units = 'D' THEN (ST_ApproximatePi * angle) / 180
               WHEN units = 'G' THEN (ST_ApproximatePi * angle) / 200
            END);
      END IF;
   END

CREATE CONSTRUCTOR METHOD ST_Angle
   (degrees INTEGER,
    minutes DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF (minutes < 0) OR
         (minutes >= 60) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument: minutes out of range';
      ELSE
         IF degrees >= 0 THEN
            RETURN NEW ST_Angle().ST_PrivateRadians(
               (ST_ApproximatePi * (degrees+(minutes/60))) / 180);
         ELSE
            RETURN NEW ST_Angle().ST_PrivateRadians(
               (ST_ApproximatePi * (degrees-(minutes/60))) / 180);
         END IF;
      END IF;
   END
```

```
CREATE CONSTRUCTOR METHOD ST_Angle
   (degrees INTEGER,
    minutes INTEGER,
    seconds DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF (minutes < 0) OR
         (minutes >= 60) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument: minutes out of range';
      ELSEIF(seconds < 0) OR
            (seconds >= 60) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument: seconds out of range';
      ELSE
         IF degrees >= 0 THEN
            RETURN NEW ST_Angle().ST_PrivateRadians(
               (ST_ApproximatePi *
                  (degrees+(minutes/60)+(seconds/3600))) / 180);
         ELSE
            RETURN NEW ST_Angle().ST_PrivateRadians(
               (ST_ApproximatePi *
                  (degrees-(minutes/60)-(seconds/3600))) / 180);
         END IF;
      END IF;
   END
```

```
CREATE CONSTRUCTOR METHOD ST_Angle
   (atpoint ST_Point,
    apoint ST_Point,
    anotherpoint ST_Point)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      -- check atpoint
      IF atpoint.ST_IsEmpty() = 1 THEN
         SIGNAL SQLSTATE '2FF17'
            SET MESSAGE_TEXT = 'empty point value';
      END IF;
      IF atpoint.ST_IsValid() = 0 THEN
         SIGNAL SQLSTATE '2FF18'
            SET MESSAGE_TEXT = 'point value not well formed';
      END IF;

      -- check apoint
      IF apoint.ST_IsEmpty() = 1 THEN
         SIGNAL SQLSTATE '2FF17'
            SET MESSAGE_TEXT = 'empty point value';
      END IF;
      IF apoint.ST_IsValid() = 0 THEN
         SIGNAL SQLSTATE '2FF18'
            SET MESSAGE_TEXT = 'point value not well formed';
      END IF;

      -- check anotherpoint
      IF anotherpoint.ST_IsEmpty() = 1 THEN
         SIGNAL SQLSTATE '2FF17'
            SET MESSAGE_TEXT = 'empty point value';
      END IF;
      IF anotherpoint.ST_IsValid() = 0 THEN
         SIGNAL SQLSTATE '2FF18'
            SET MESSAGE_TEXT = 'point value not well formed';
      END IF;

      -- check if points are coincident
      IF atpoint.ST_Equals(apoint) = 1 THEN
         -- points are co-located so direction is not defined
         SIGNAL SQLSTATE '2FF19'
            SET MESSAGE_TEXT = 'points are equal';
      END IF;

      IF atpoint.ST_Equals(anotherpoint) = 1 THEN
         -- points are co-located so direction is not defined
         SIGNAL SQLSTATE '2FF19'
            SET MESSAGE_TEXT = 'points are equal';
      END IF;
      --
      -- See Description
      --
   END
```

```
CREATE CONSTRUCTOR METHOD ST_Angle
   (adirection ST_Direction,
    anotherdirection ST_Direction)
   RETURNS ST_Angle
   FOR ST_Angle
   RETURN NEW ST_Angle(
      CASE
         WHEN (anotherdirection.ST_Radians()-adirection.ST_Radians()) <
            (adirection.ST_Radians()-anotherdirection.ST_Radians()) THEN
            anotherdirection.ST_Radians()-adirection.ST_Radians()
         ELSE
            adirection.ST_Radians()-anotherdirection.ST_Radians()
      END)

CREATE CONSTRUCTOR METHOD ST_Angle
   (aline ST_LineString,
    anotherline ST_LineString)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      -- check if aline is valid
      IF aline.ST_NumPoints() <> 2 THEN
         -- not a line
         SIGNAL SQLSTATE '2FF20'
            SET MESSAGE_TEXT = 'linestring is not a line';
      ELSEIF aline.ST_IsClosed() = 1 THEN
         -- not a line
         SIGNAL SQLSTATE '2FF21'
            SET MESSAGE_TEXT = 'degenerate line has no direction';
      END IF;

      -- check if anotherline is valid
      IF anotherline.ST_NumPoints() <> 2 THEN
         -- not a line
         SIGNAL SQLSTATE '2FF20'
            SET MESSAGE_TEXT = 'linestring is not a line';
      ELSEIF anotherline.ST_IsClosed() = 1 THEN
         -- not a line
         SIGNAL SQLSTATE '2FF21'
            SET MESSAGE_TEXT = 'degenerate line has no direction';
      END IF;
      RETURN NEW ST_Angle(
         NEW ST_Direction(aline), NEW ST_Direction(anotherline));
   END

CREATE CONSTRUCTOR METHOD ST_Angle
   (awkt CHARACTER VARYING(ST_MaxAngleAsText))
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Angle
   (agml CHARACTER LARGE OBJECT(ST_MaxAngleAsGML))
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      --
      -- See Description
```



```
    --
  END
```

Definitional Rules

1) *ST_MaxAngleAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Angle* value.
2) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.

Description

1) The method *ST_Angle(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *angle*.
2) The null-call type-preserving SQL-invoked constructor method *ST_Angle(DOUBLE PRECISION)* returns an *ST_Angle* value with the attribute *ST_PrivateRadians* set to *angle*.
3) The method *ST_Angle(CHARACTER, DOUBLE PRECISION)* takes the following input parameters:
   a) a CHARACTER value *units*,
   b) a DOUBLE PRECISION value *angle*.
4) Let *IND* be the CHARACTER value specified by *units*.
5) For the null-call type-preserving SQL-invoked constructor method *ST_Angle(CHARACTER, DOUBLE PRECISION)* return an *ST_Angle* value with:

   Case:

   a) If IND is not 'R', 'D' or 'G', then an exception condition is raised: *SQL/MM Spatial exception – invalid argument: invalid angle units*.
   b) Otherwise, return an *ST_Angle* value with:

      Case:

      i) If *IND* is equal to 'R', then the attribute *ST_PrivateRadians* set to *angle*.
      i) If *IND* is equal to 'D', then the attribute *ST_PrivateRadians* set to *(π * angle) / 180*.
      iii) If *IND* is equal to 'G', then the attribute *ST_PrivateRadians* set to *(π * angle) / 200*.
6) The method *ST_Angle(INTEGER, DOUBLE PRECISION)* takes the following input parameters:
   a) an INTEGER value *degrees*,
   b) a DOUBLE PRECISION value *minutes*.
7) Let *D* be the INTEGER value specified by *degrees*, and *M* the DOUBLE PRECISION value specified by *minutes*.
8) If *M* is less than 0 or *M* is greater than or equal to 60, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument: minutes out of range*.
9) For the null-call type-preserving SQL-invoked constructor method *ST_Angle(INTEGER, DOUBLE PRECISION)* return an *ST_Angle* value with:

   Case:

   a) If *D* >= 0 (zero), then the attribute *ST_PrivateRadians* set to the angle measurement represented by the combination of *degrees* and *minutes,* expressed in radians, equal to:

      $(\pi * (D + M / 60)) / 180$
   b) Otherwise, the attribute *ST_PrivateRadians* set to the angle measurement represented by the combination of *degrees* and *minutes*, expressed in radians, equal to:

      $(\pi * (D - M / 60)) / 180$
10) The method *ST_Angle(INTEGER, INTEGER, DOUBLE PRECISION)* takes the following input parameters:

a) an INTEGER value *degrees*,

b) an INTEGER value *minutes*,

c) a DOUBLE PRECISION value *seconds*.

11) Let *D* be the INTEGER value specified by *degrees*, *M* the INTEGER value specified by *minutes*, and *S* the DOUBLE PRECISION value specified by *seconds*.

12) If *M* is less than 0 or *M* is greater than or equal to 60, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument: minutes out of range*.

13) If *S* is less than 0 or *S* is greater than or equal to 60, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument: seconds out of range*.

14) For the null-call type-preserving SQL-invoked constructor method *ST_Angle(INTEGER, INTEGER, DOUBLE PRECISION)* return an *ST_Angle* value with:

Case:

a) If *D* >= 0 (zero), then the attribute *ST_PrivateRadians* set to the angle measurement represented by the combination of *degrees*, *minutes*, and *seconds*, expressed in radians, equal to:

$$(\pi * (D + M / 60 + S / 3600)) / 180$$

b) Otherwise, the attribute *ST_PrivateRadians* set to the angle measurement represented by the combination of *degrees*, *minutes*, and *seconds*, expressed in radians, equal to:

$$(\pi * (D - M / 60 - S / 3600)) / 180$$

15) The method *ST_Angle(ST_Point, ST_Point, ST_Point)* takes the following input parameters:

a) an *ST_Point* value *atpoint*,

b) an *ST_Point* value *apoint*,

c) an *ST_Point* value *anotherpoint*.

16) Let *D1* be the direction from *atpoint* to *apoint* and let *D2* be the direction from *atpoint* to *anotherpoint*.

17) For the null-call type-preserving SQL-invoked constructor method *ST_Angle(ST_Point, ST_Point, ST_Point)*:

Case:

a) If *atpoint* is the empty set, then an exception condition is raised: *SQL/MM Spatial exception – empty point value*.

b) If *atpoint* is not well formed, then an exception condition is raised: *SQL/MM Spatial exception – point value not well formed*.

c) If *apoint* is the empty set, then an exception condition is raised: *SQL/MM Spatial exception – empty point value*.

d) If *apoint* is not well formed, then an exception condition is raised: *SQL/MM Spatial exception – point value not well formed*.

e) If *anotherpoint* is the empty set, then an exception condition is raised: *SQL/MM Spatial exception – empty point value*.

f) If *anotherpoint* is not well formed, then an exception condition is raised: *SQL/MM Spatial exception – point value not well formed*.

g) If *atpoint* is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – points are equal*.

h) If *atpoint* is equal to *anotherpoint*, then an exception condition is raised: *SQL/MM Spatial exception – points are equal*.

i) Otherwise, return an ST_Angle value with the attribute ST_PrivateRadians set to the lesser of:

i) the clockwise angle measured in radians from *D1* to *D2*, or

ii) the clockwise angle measured in radians from *D2* to *D1*.

18) The method *ST_Angle(ST_Direction, ST_Direction)* takes the following input parameters:
    a) an *ST_Direction* value *adirection*,
    b) an *ST_Direction* value *anotherdirection*.
19) For the null-call type-preserving SQL-invoked constructor method *ST_Angle(ST_Direction, ST_Direction)* return an *ST_Angle* value with the attribute *ST_PrivateRadians* set to the lesser of:
    a) the clockwise angle measured in radians from *adirection* to *anotherdirection*, or
    b) the clockwise angle measured in radians from *anotherdirection* to *adirection*.
20) The method *ST_Angle(ST_LineString, ST_LineString)* takes the following input parameters:
    a) an *ST_LineString* value *aline*,
    b) an *ST_LineString* value *anotherline*.
21) Let *P1* be the point value returned by *aline.ST_StartPoint()* and *P2* be the point value returned by *aline.ST_EndPoint()*. Let *D1* be the direction from *P1* to *P2*.
22) Let *P3* be the point value returned by *anotherline.ST_StartPoint()* and *P4* be the point value returned by *anotherline.ST_EndPoint()* Then let *D2* be the direction from *P3* to *P4*.
23) For the null-call type-preserving SQL-invoked constructor method *ST_Angle(ST_LineString, ST_LineString)*:

    Case:
    a) If *aline* is not a line, then an exception condition is raised: *SQL/MM Spatial exception – linestring is not a line*.
    b) If *aline* is closed, then an exception condition is raised: *SQL/MM Spatial exception – degenerate line has no direction*.
    c) If *anotherline* is not a line, then an exception condition is raised: *SQL/MM Spatial exception – linestring is not a line*.
    d) If *anotherline* is closed, then an exception condition is raised: *SQL/MM Spatial exception – degenerate line has no direction*.
    e) Otherwise, return an *ST_Angle* value with the attribute *ST_PrivateRadians* set to the lesser of:
        i) the clockwise angle measured in radians from *D1* to *D2*, or
        ii) the clockwise angle measured in radians from *D2* to *D1*
24) The method *ST_Angle(CHARACTER VARYING)* takes the following input parameters:
    a) a CHARACTER VARYING value *awkt*.
25) The null-call type-preserving SQL-invoked constructor method *ST_Angle(CHARACTER VARYING)* returns the result of the value expression: *ST_AngleFromText(awkt)*.
26) The method *ST_Angle(CHARACTER LARGE OBJECT)* takes the following input parameters:
    a) a CHARACTER LARGE OBJECT value *agml.*
27) The null-call type-preserving SQL-invoked constructor method *ST_Angle(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_AngleFromGML(agml)*.

16.1.3 ST_Radians Methods

Purpose

Observe and mutate the radians attribute of an ST_Angle value.

Definition

```
CREATE METHOD ST_Radians()
   RETURNS DOUBLE PRECISION
   FOR ST_Angle
   RETURN SELF.ST_PrivateRadians

CREATE METHOD ST_Radians
   (radians DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF radians IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateRadians(radians)
            END;
      END IF;
   END
```

Description

1) The method *ST_Radians()* has no input parameters.

2) The null-call method *ST_Radians()* returns the value of the *ST_PrivateRadians* attribute.

3) The method *ST_Radians(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *radians*.

4) For the type-preserving method *ST_Radians(DOUBLE PRECISION)*:

   Case:

   a) If *radians* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateRadians(radians)*.

16.1.4 ST_Degrees Methods

Purpose

Observe and mutate the radians attribute of an ST_Angle value using decimal degrees.

Definition

```
CREATE METHOD ST_Degrees()
   RETURNS DOUBLE PRECISION
   FOR ST_Angle
   RETURN (SELF.ST_PrivateRadians / ST_ApproximatePi) * 180

CREATE METHOD ST_Degrees
   (degrees DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF degrees IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateRadians(
                     (ST_ApproximatePi * degrees) / 180)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_ApproximatePi* is the implementation-defined meta-variable representing π.

Description

1) The method *ST_Degrees()* has no input parameters.
2) The null-call method *ST_Degrees()* returns the value of the *ST_PrivateRadians* attribute converted to decimal degrees.
3) The method *ST_Degrees(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *degrees*.
4) For the type-preserving method *ST_Degrees(DOUBLE PRECISION)*:

   Case:

   a) If *degrees* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression:
      *SELF.ST_PrivateRadians((ST_ApproximatePi * degrees) / 180)*.

16.1.5 ST_DegreeComponent Method

Purpose

Observe the INTEGER degrees part of the degrees, minutes, and seconds representation of the radians attribute of an ST_Angle value.

Definition

```
CREATE METHOD ST_DegreeComponent()
   RETURNS INTEGER
   FOR ST_Angle
   RETURN FLOOR(SELF.ST_Degrees())
```

Description

1) The method *ST_DegreeComponent()* has no input parameters.
2) The null-call method *ST_DegreeComponent()* returns the INTEGER degree part of the degrees, minutes, and seconds representation of the value of the *ST_PrivateRadians* attribute.

16.1.6 ST_MinuteComponent Method

Purpose

Observe the INTEGER minutes part of the degrees, minutes, and seconds representation of the radians attribute of an ST_Angle value.

Definition

```
CREATE METHOD ST_MinuteComponent()
   RETURNS INTEGER
   FOR ST_Angle
   RETURN FLOOR(ABS((SELF.ST_Degrees()-SELF.ST_DegreeComponent())*60))
```

Description

1) The method *ST_MinuteComponent()* has no input parameters.

2) The null-call method *ST_MinuteComponent()* returns the INTEGER minutes part of the degrees, minutes, and seconds representation of the value of the *ST_PrivateRadians* attribute.

16.1.7 ST_SecondComponent Method

Purpose

Observe the DOUBLE PRECISION seconds part of the degrees, minutes, and seconds representation of the radians attribute of an ST_Angle value.

Definition

```
CREATE METHOD ST_SecondComponent()
   RETURNS DOUBLE PRECISION
   FOR ST_Angle
   RETURN
      CASE
         WHEN SELF.ST_Degrees() >= 0 THEN
            SELF.ST_Degrees()-
               (SELF.ST_DegreeComponent()+
                (SELF.ST_MinuteComponent()/60)
               )*3600
         ELSE
            SELF.ST_Degrees()-
               (SELF.ST_DegreeComponent()-
                (SELF.ST_MinuteComponent()/60)
               )*3600
      END
```

Description

1) The method *ST_SecondComponent()* has no input parameters.
2) The null-call method *ST_SecondComponent()* returns the DOUBLE PRECISION seconds part of the degrees, minutes, and seconds representation of the value of the *ST_PrivateRadians* attribute.

16.1.8 ST_String Methods

Purpose

Observe and mutate the radians attribute of an ST_Angle value using a space separated character string of degrees, minutes, and seconds.

Definition

```
CREATE METHOD ST_String()
   RETURNS CHARACTER VARYING(ST_MaxAngleString)
   FOR ST_Angle
   RETURN SELF.ST_String(2)

CREATE METHOD ST_String
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxAngleString)
   FOR ST_Angle
   BEGIN
      IF (numdecdigits < 0) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument: number of digits is
            negative';
      ELSE
         RETURN CAST (SELF.ST_DegreeComponent() AS
               CHARACTER VARYING(ST_MaxAngleString-(7+numdecdigits))) ||
            ' ' ||
            CAST (SELF.ST_MinuteComponent() AS CHARACTER VARYING(2)) ||
            ' ' ||
            CAST (ROUND(SELF.ST_SecondComponent(), numdecdigits)
               AS CHARACTER VARYING(3+numdecdigits));
      END IF;
   END

CREATE METHOD ST_String
   (dms CHARACTER VARYING(ST_MaxAngleString))
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      DECLARE degrees DOUBLE PRECISION;

      IF dms IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  --
                  -- SET degrees = !! See Description
                  --
                  SELF.ST_PrivateRadians(
                     (ST_ApproximatePi * degrees)/180)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_ApproximatePi* is the implementation-defined meta-variable representing π.

2) *ST_MaxAngleString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of the *ST_PrivateRadians* attribute of an *ST_Angle* value.

Description

1) The method *ST_String()* has no input parameters.
2) The null-call method *ST_String()* returns the value of the *ST_PrivateRadians* attribute expressed as a space separated string of degrees, minutes, and seconds. The value of seconds shall be rounded or truncated to 2 decimal digits. The choice of whether to truncate or round is implementation-defined.
3) The method *ST_String(INTEGER)* takes the following input parameters:
   a) an INTEGER value *numdecdigits*.
4) The null-call method *ST_String(INTEGER)* returns the value of the *ST_PrivateRadians* attribute expressed as a space separated string of degrees, minutes, and seconds. The value of seconds shall be rounded or truncated to the number of decimal places indicated by the lesser of

   Case:

   a) *numdecdigits*
   b) *ST_MaxAngleString* minus (7 plus the number of digits needed to express the degrees part).

   The choice of whether to truncate or round is implementation-defined.
5) Case:
   a) If *numdecdigits* is less than 0 , then an exception condition is raised: *SQL/MM Spatial exception – invalid argument: number of digits is negative*.
   b) Otherwise, the maximum measure value of *numdecdigits* is implementation-defined.
6) The method *ST_String(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *dms*.
7) For the type-preserving method *ST_String(CHARACTER VARYING)*:

   Case:

   a) If *dms* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise:
      i) Let *DEGREES* equal the DOUBLE PRECISION value obtained by converting the degrees, minutes, and seconds representation of an angle represented by *dms*, into an equivalent decimal degrees representation.
      ii) Return the result of the value expression: *SELF.ST_PrivateRadians((ST_ApproximatePi * DEGREES) / 180)*.

16.1.9 ST_Gradians Methods

Purpose

Observe and mutate the radians attribute of an ST_Angle value using gradians.

Definition

```
CREATE METHOD ST_Gradians()
   RETURNS DOUBLE PRECISION
   FOR ST_Angle
   RETURN (SELF.ST_PrivateRadians / ST_ApproximatePi) * 200

CREATE METHOD ST_Gradians
   (gradians DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF gradians IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateRadians(
                     (ST_ApproximatePi * gradians) / 200)
            END;
      END IF;
   END
```

Definitional Rules

1) *ST_ApproximatePi* is the implementation-defined meta-variable representing π.

Description

1) The method *ST_Gradians()* has no input parameters.

2) The null-call method *ST_Gradians()* returns the value of the *ST_PrivateRadians* attribute converted to gradians.

3) The method *ST_Gradians(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *gradians*.

4) For the type-preserving method *ST_Gradians(DOUBLE PRECISION)*:

   Case:

   a) If *gradians* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateRadians((ST_ApproximatePi * gradians) / 200)*.

16.1.10 ST_Add Method

Purpose

Add the value of an angle to the ST_Angle value.

Definition

```
CREATE METHOD ST_Add
   (anangle ST_Angle)
   RETURNS ST_Angle
   FOR ST_Angle
   RETURN SELF.ST_PrivateRadians(
      SELF.ST_PrivateRadians+anangle.ST_PrivateRadians)
```

Description

1) The method *ST_Add(ST_Angle)* takes the following input parameters:

   a) an *ST_Angle* value *anangle*.

2) The null-call type-preserving method *ST_Add(ST_Angle)* returns the value of SELF with the *SELF.ST_PrivateRadians* attribute value set to its original value plus the value of *anangle.ST_PrivateRadians*.

16.1.11 ST_Subtract Method

Purpose

Subtract the value of an angle from the ST_Angle value.

Definition

```
CREATE METHOD ST_Subtract
   (anangle ST_Angle)
   RETURNS ST_Angle
   FOR ST_Angle
   RETURN SELF.ST_PrivateRadians(
      SELF.ST_PrivateRadians-anangle.ST_PrivateRadians)
```

Description

1) The method *ST_Subtract(ST_Angle)* takes the following input parameters:
   a) an *ST_Angle* value *anangle*.
2) The null-call type-preserving method *ST_Subtract(ST_Angle)* returns the value of SELF with the *SELF.ST_PrivateRadians* attribute value set to its original value minus the value of *anangle.ST_PrivateRadians*.

16.1.12 ST_Multiply Method

Purpose

Multiply the ST_Angle value by a numeric value.

Definition

```
CREATE METHOD ST_Multiply
   (afactor DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   RETURN SELF.ST_PrivateRadians(SELF.ST_PrivateRadians*afactor)
```

Description

1) The method *ST_Multiply(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *afactor*.

2) The null-call type-preserving method *ST_Multiply(ST_Angle)* returns the value of SELF with the *SELF.ST_PrivateRadians* attribute value set to its original value multiplied by *afactor*.

16.1.13 ST_Divide Method

Purpose

Divide the ST_Angle value by a non-zero, numeric value.

Definition

```
CREATE METHOD ST_Divide
   (adivisor DOUBLE PRECISION)
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      IF adivisor = 0 THEN
         SIGNAL SQLSTATE '2FF13'
         SET MESSAGE_TEXT = 'attempted division by zero';
      END IF;
      RETURN SELF.ST_PrivateRadians(SELF.ST_PrivateRadians/adivisor);
   END
```

Description

1) The method *ST_Divide(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *adivisor*.
2) For the null-call type-preserving method *ST_Divide(ST_Angle)*:

   Case:

   a) If *adivisor* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – attempted division by zero*.
   b) Otherwise, return the value of SELF with the *SELF.ST_PrivateRadians* attribute value set to its original value divided by *adivisor*.

16.1.14 ST_AsText Method

Purpose

Return the well-known text representation of an ST_Angle value.

Definition

```
CREATE METHOD ST_AsText()
   RETURNS CHARACTER VARYING(ST_MaxAngleAsText)
   FOR ST_Angle
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxAngleAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Angle* value.

Description

1) The method *ST_AsText()* has no input parameters.
2) The null-call method *ST_AsText()* returns a CHARACTER VARYING value containing the well-known text representation of SELF. Values shall be produced in the BNF for <angle text representation>.

16.1.15 ST_GMLToSQL Method

Purpose

Return an ST_Angle value for a given GML representation.

Definition

```
CREATE METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxAngleAsGML))
   RETURNS ST_Angle
   FOR ST_Angle
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.

Description

1) The method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The parameter *agml* is the GML Angle representation of an *ST_Angle* value. If *agml* does not contain an Angle XML element, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
3) The null-call method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* returns an *ST_Angle* value represented by *agml*.

16.1.16 ST_AsGML Method

Purpose

Return the GML representation of an ST_Angle value.

Definition

```
CREATE METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxAngleAsGML)
   FOR ST_Angle
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.

Description

1) The method *ST_AsGML()* has no input parameters.

2) The null-call method *ST_AsGML()* returns a CHARACTER LARGE OBJECT value containing a GML representation. The instantiable type of ST_Angle is mapped to an Angle XML element in the GML representation.

16.1.17 ST_AngleFromText Function

Purpose

Return an ST_Angle value which is transformed from a CHARACTER VARYING value that represents the well-known text representation of an ST_Angle value.

Definition

```
CREATE FUNCTION ST_AngleFromText
   (awkt CHARACTER VARYING(ST_MaxAngleAsText))
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxAngleAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Angle* value.

Description

1) The function *ST_AngleFromText(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *awkt*.
2) For the null-call function *ST_AngleFromText(CHARACTER VARYING)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Angle* value.

      If *awkt* is not producible in the BNF for <angle text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_AngleFromText(awkt) AS ST_Angle)*.

16.1.18 ST_AngleFromGML Function

Purpose

Return an ST_Angle value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Angle representation of an ST_Angle value.

Definition

```
CREATE FUNCTION ST_AngleFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxAngleAsGML))
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.

Description

1) The function *ST_AngleFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_AngleFromGML(CHARACTER LARGE OBJECT)*:
   a) If the parameter *agml* does not contain an Angle XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_AngleFromGML(agml) AS ST_Angle)*.

16.1.19 ST_Angle Ordering Definition

Purpose

Define ordering for ST_Angle.

Definition

```
CREATE FUNCTION ST_OrderingCompare
   (anangle ST_Angle,
    anotherangle ST_Angle)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN
      CASE
         WHEN anangle.ST_PrivateRadians <
               anotherangle.ST_PrivateRadians THEN
            -1
         WHEN anangle.ST_PrivateRadians >
               anotherangle.ST_PrivateRadians THEN
            1
         ELSE
            0
      END
```

```
CREATE ORDERING FOR ST_Angle
   ORDER FULL BY RELATIVE
      WITH FUNCTION ST_OrderingCompare(ST_Angle, ST_Angle)
```

Description

1) The function *ST_OrderingCompare(ST_Angle, ST_Angle)* takes the following input parameters:
   a) an *ST_Angle* value *anangle*,
   b) an *ST_Angle* value *anotherangle*.
2) For the null-call function *ST_OrderingCompare(ST_Angle, ST_Angle)*:

   Case:
   a) If *anangle.ST_PrivateRadians* < *anotherangle.ST_PrivateRadians*, then return -1.
   b) If *anangle.ST_PrivateRadians* > *anotherangle.ST_PrivateRadians*, then return 1 (one).
   c) Otherwise, return 0 (zero).
3) Use the function *ST_OrderingCompare(ST_Angle, ST_Angle)* to define ordering for the *ST_Angle* type.

16.1.20 SQL Transform Functions

Purpose

Define SQL transform functions for the ST_Angle type.

Definition

```
CREATE TRANSFORM FOR ST_Angle
   ST_WellKnownText
      (TO SQL WITH METHOD ST_Angle(CHARACTER VARYING(ST_MaxAngleAsText)),
       FROM SQL WITH METHOD ST_AsText())
   ST_WellKnownBinary
      (TO SQL WITH METHOD ST_Angle(DOUBLE PRECISION),
       FROM SQL WITH METHOD ST_Radians())
   ST_GML
      (TO SQL WITH METHOD ST_GMLToSQL
      (CHARACTER LARGE OBJECT(ST_MaxAngleAsGML)),
       FROM SQL WITH METHOD ST_AsGML())
```

Definitional Rules

1) *ST_MaxAngleAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Angle* value.
2) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.

Description

1) Use the method *ST_Angle(CHARACTER VARYING)* and the method *ST_AsText()* to define the transform group *ST_WellKnownText*.
2) Use the method *ST_Angle(DOUBLE PRECISION)* and the method *ST_Radians()* to define the transform group *ST_WellKnownBinary*.
3) Use the method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsGML()* to define the transform group *ST_GML*.

16.1.21 <angle text representation>

Purpose

This subclause contains the definition of the <well-known text representation> of an ST_Angle value.

Description

1) The well-known text representation of an *ST_Angle* value is defined by the following BNF for <angle text representation>:

```
<angle text representation> ::=
     ANGLE <angle text>

<angle text> ::=
     DEGREES <left paren> <degrees> <right paren>
   | GRADIANS <left paren> <gradians> <right paren>
   | RADIANS <left paren> <radians> <right paren>

<degrees> ::=
     <number>

<gradians> ::=
     <number>

<radians> ::=
     <number>

<left paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<number> ::=
     <exact numeric literal>
   | <approximate numeric literal>

<exact numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<approximate numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075
```

a) <angle text representation> is the well-known text representation for an *ST_Angle* value that is produced by <angle text>.

b) <angle text> produces the *ST_Angle* value from <degrees>, <gradians>, or <radians>.

c) Case:

   i) Let *DEGREES* be the DOUBLE PRECISION value specified by <degrees>. <degrees> produces an *ST_Angle* value as the result of the value expression: *NEW ST_Angle*('D', *DEGREES*).

   ii) Let *GRADIANS* be the DOUBLE PRECISION value specified by <gradians>. <gradians> produces an *ST_Angle* value as the result of the value expression: *NEW ST_Angle*('G', *GRADIANS*).

   iii) Let *RADIANS* be the DOUBLE PRECISION value specified by <radians>. <radians> produces an *ST_Angle* value as the result of the value expression: *NEW ST_Angle*('R', *RADIANS*).

d) The list of keywords is: ANGLE, DEGREES, GRADIANS, and RADIANS.

16.2 ST_Direction Type and Routines

16.2.1 ST_Direction Type

Purpose

The ST_Direction type is used to express direction, either as an azimuth or bearing.

Definition

```
CREATE TYPE ST_Direction
   AS (
      ST_PrivateAngleNAzimuth ST_Angle DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Direction
      (direction DOUBLE PRECISION)
      RETURNS ST_Direction
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Direction
      (northsouth CHARACTER(1),
       bearingangle ST_Angle,
       eastwest CHARACTER(1))
      RETURNS ST_Direction
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Direction
      (northsouth CHARACTER(1),
       azimuthangle ST_Angle)
      RETURNS ST_Direction
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Direction
      (frompoint ST_Point,
       topoint ST_Point)
      RETURNS ST_Direction
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Direction
   (aline ST_LineString)
   RETURNS ST_Direction
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Direction
   (awkt CHARACTER VARYING(ST_MaxDirectionAsText))
   RETURNS ST_Direction
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Direction
   (agml CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML))
   RETURNS ST_Direction
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Radians()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AngleNAzimuth()
   RETURNS ST_Angle
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AngleNAzimuth
   (nazimuthangle ST_Angle)
   RETURNS ST_Direction
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_AsText()
   RETURNS CHARACTER VARYING(ST_MaxDirectionAsText)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_RadianBearing
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DegreesBearing
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DMSBearing
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_RadianNAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DegreesNAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DMSNAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_RadianSAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_DegreesSAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_DMSSAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AddAngle
   (anangle ST_Angle)
   RETURNS ST_Direction
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SubtractAngle
   (anangle ST_Angle)
   RETURNS ST_Direction
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML))
   RETURNS ST_Direction
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.
2) *ST_MaxDirectionAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Direction* value.
3) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.
4) The attribute *ST_PrivateAngleNAzimuth* is not for public use. There are no GRANT statements granting EXECUTE privilege to the observer or mutator method for *ST_PrivateAngleNAzimuth*.

Description

1) The *ST_Direction* type provides for public use:
   a) a method *ST_Direction(DOUBLE PRECISION)*,
   b) a method *ST_Direction(CHARACTER, ST_Angle, CHARACTER)*,
   c) a method *ST_Direction(CHARACTER, ST_Angle)*,
   d) a method *ST_Direction(ST_Point, ST_Point)*,
   e) a method *ST_Direction(ST_LineString)*,
   f) a method *ST_Direction(CHARACTER VARYING)*,
   g) a method *ST_Direction(CHARACTER LARGE OBJECT)*,
   h) a method *ST_Radians()*,
   i) a method *ST_AngleNAzimuth()*,
   j) a method *ST_AngleNAzimuth(ST_Angle)*,
   k) a method *ST_GMLToSQL(CHARACTER LARGE OBJECT)*,
   l) a method *ST_AsGML()*,
   m) a method *ST_AsText()*,
   n) a method *ST_RadianBearing(INTEGER)*,
   o) a method *ST_DegreesBearing(INTEGER)*,
   p) a method *ST_DMSBearing(INTEGER)*,
   q) a method *ST_RadianNAzimuth(INTEGER)*,
   r) a method *ST_DegreesNAzimuth(INTEGER)*,
   s) a method *ST_DMSNAzimuth(INTEGER)*,
   t) a method *ST_RadianSAzimuth(INTEGER)*,
   u) a method *ST_DegreesSAzimuth(INTEGER)*,
   v) a method *ST_DMSSAzimuth(INTEGER)*,
   w) a method *ST_AddAngle(ST_Angle)*,
   x) a method *ST_SubtractAngle(ST_Angle)*,
   y) an ordering function *ST_OrderingCompare(ST_Direction, ST_Direction)*,
   z) an SQL Transform group *ST_WellKnownText*,
   aa) an SQL Transform group *ST_WellKnownBinary*,
   ab) an SQL Transform group *ST_GML*,
   ac) a function *ST_DirectionFrmTxt(CHARACTER VARYING)*,
   ad) a function *ST_DirectionFrmGML(CHARACTER LARGE OBJECT)*.
2) The attribute *ST_PrivateAngleNAzimuth* contains the angle measured clockwise from True North.
3) The value of the angle in *ST_PrivateAngleNAzimuth* shall be greater than or equal to 0 (zero) and less than 360 degrees (or 2π radians or 400 gradians). Methods mutating the value of an *ST_Direction* (for example, *ST_AddAngle*) shall convert the resultant to be within this range.

16.2.2 ST_Direction Methods

Purpose

Return a specified ST_Direction value.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Direction
   (direction DOUBLE PRECISION)
   RETURNS ST_Direction
   FOR ST_Direction
   RETURN NEW ST_Direction().
      ST_PrivateAngleNAzimuth(NEW ST_Angle(direction))

CREATE CONSTRUCTOR METHOD ST_Direction
   (northsouth CHARACTER(1),
    bearingangle ST_Angle,
    eastwest CHARACTER(1))
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Direction
   (northsouth CHARACTER(1),
    azimuthangle ST_Angle)
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      --
      -- See Description
      --
   END
```

```
CREATE CONSTRUCTOR METHOD ST_Direction
   (frompoint ST_Point,
    topoint ST_Point)
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      -- check frompoint
      IF frompoint.ST_IsEmpty() = 1 THEN
         SIGNAL SQLSTATE '2FF17'
            SET MESSAGE_TEXT = 'empty point value';
      END IF;
      IF frompoint.ST_IsValid() = 0 THEN
         SIGNAL SQLSTATE '2FF18'
            SET MESSAGE_TEXT = 'point value not well formed';
      END IF;

      -- check topoint
      IF topoint.ST_IsEmpty() = 1 THEN
         SIGNAL SQLSTATE '2FF17'
            SET MESSAGE_TEXT = 'empty point value';
      END IF;
      IF topoint.ST_IsValid() = 0 THEN
         SIGNAL SQLSTATE '2FF18'
            SET MESSAGE_TEXT = 'point value not well formed';
      END IF;

      -- check if points are coincident
      IF frompoint.ST_Equals(topoint) = 1 THEN
         -- points are co-located so direction is not defined
         SIGNAL SQLSTATE '2FF19'
            SET MESSAGE_TEXT = 'points are equal';
      END IF;
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Direction
   (aline ST_LineString)
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      -- check if aline is valid
      IF aline.ST_NumPoints() <> 2 THEN
         -- not a line
         SIGNAL SQLSTATE '2FF20'
            SET MESSAGE_TEXT = 'linestring is not a line';
      ELSEIF aline.ST_IsClosed() = 1 THEN
         -- not a line
         SIGNAL SQLSTATE '2FF21'
            SET MESSAGE_TEXT = 'degenerate line has no direction';
      END IF;
      RETURN NEW ST_Direction(aline.ST_StartPoint(), aline.ST_EndPoint());
   END
```

```
CREATE CONSTRUCTOR METHOD ST_Direction
   (awkt CHARACTER VARYING(ST_MaxDirectionAsText))
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Direction
   (agml CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML))
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxDirectionAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Direction* value.
2) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.

Description

1) The method *ST_Direction(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *direction*, measured in radians.
2) For the null-call type-preserving SQL-invoked constructor method *ST_Direction(DOUBLE PRECISION)* returns:

   Case:

   a) If *direction* is less than 0 (zero) radians or *direction* is greater than or equal to (2π) radians, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   b) Otherwise, return an *ST_Direction* value with the attribute *ST_PrivateAngleNAzimuth* set to *NEW ST_Angle(direction*).
3) The method *ST_Direction(CHARACTER, ST_Angle, CHARACTER)* takes the following input parameters:
   a) a CHARACTER value *northsouth*,
   b) an *ST_Angle* value *bearingangle*,
   c) a CHARACTER value *eastwest*.
4) For the null-call type-preserving SQL-invoked constructor method *ST_Direction(CHARACTER, ST_Angle, CHARACTER)*:
   a) If *northsouth* is not 'N' (for North) or 'S' (for South), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   b) If *bearingangle.ST_Radians()* is less than 0 (zero) radians or *bearingangle.ST_Radians()* is greater than (π/2) radians, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   c) If *eastwest* is not 'E' (for East) or 'W' (for West), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   d) Otherwise, return an *ST_Direction* value with the attribute *ST_PrivateAngleNAzimuth* set to the *ST_Angle* value which, when measured clockwise from True North, specifies a direction equivalent to the bearing specified by *northsouth*, *bearingangle,* and *eastwest*.

5) The method *ST_Direction(CHARACTER, ST_Angle)* takes the following input parameters:
   a) a CHARACTER value *northsouth*,
   b) an *ST_Angle* value *azimuthangle*.
6) Let *NS* be the CHARACTER value specified by *northsouth*.
7) For the null-call type-preserving SQL-invoked constructor method *ST_Direction(CHARACTER, ST_Angle)*:

   Case:
   a) If *NS* is not 'N' (for North) or 'S' (for South), then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   b) If *azimuthangle.ST_Radians()* is less than 0 (zero) radians or *azimuthangle.ST_Radians()* is greater than or equal to (2π) radians, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
   c) Otherwise, return an *ST_Direction* value with:

      Case:
      i) If *NS* is equal to 'N', then the attribute *ST_PrivateAngleNAzimuth* set to *azimuthangle*.
      ii) If *NS* is equal to 'S' and 0 (zero) <= *azimuthangle.ST_Degrees()* < 180, then the attribute *ST_PrivateAngleNAzimuth* set to *azimuthangle.ST_Add*(*NEW ST_Angle('D',180))*.
      iii) Otherwise, the attribute *ST_PrivateAngleNAzimuth* set to *azimuthangle.ST_Subtract*(*NEW ST_Angle('D',180))*.
8) The method *ST_Direction(ST_Point, ST_Point)* takes the following input parameters:
   a) an *ST_Point* value *frompoint,*
   b) an *ST_Point* value *topoint*.
9) For the null-call type-preserving SQL-invoked constructor method *ST_Direction(ST_Point, ST_Point)*:

   Case:
   a) If *frompoint* is the empty set, then an exception condition is raised: *SQL/MM Spatial exception – empty point value*.
   b) If *frompoint* is not well formed, then an exception condition is raised: *SQL/MM Spatial exception – point value not well formed*.
   c) If *topoint* is the empty set, then an exception condition is raised: *SQL/MM Spatial exception – empty point value*.
   d) If *topoint* is not well formed, then an exception condition is raised: *SQL/MM Spatial exception – point value not well formed*.
   e) If *frompoint* is equal to *topoint*, then an exception condition is raised: *SQL/MM Spatial exception – points are equal*.
   f) Otherwise, return an *ST_Direction* value with:
      i) the attribute *ST_PrivateAngleNAzimuth* set to the *ST_Angle* value which, when measured clockwise from True North, specifies the North azimuth direction from the *frompoint* to the *topoint*.
10) The method *ST_Direction(ST_LineString)* takes the following input parameters:
    a) an *ST_LineString* value *aline*.
11) Let *P1* be the point value returned by *aline.ST_StartPoint()* and *P2* be the point value returned by *aline.ST_EndPoint()*.
12) For the null-call type-preserving SQL-invoked constructor method *ST_Direction(ST_LineString)*:

    Case:

a) If *aline* is not a line, then an exception condition is raised: *SQL/MM Spatial exception – linestring is not a line*.

b) If *aline* is closed, then an exception condition is raised: *SQL/MM Spatial exception – degenerate line has no direction*.

c) Otherwise, return the result of the value expression: *NEW ST_Direction(P1, P2)*.

13) The method *ST_Direction(CHARACTER VARYING)* takes the following input parameters:

a) a CHARACTER VARYING value *awkt.*

14) The null-call type-preserving SQL-invoked constructor method *ST_Direction(CHARACTER VARYING)* returns the result of the value expression: *ST_DirectionFrmTxt(awkt)*.

15) The method *ST_Direction(CHARACTER LARGE OBJECT)* takes the following input parameters:

a) a CHARACTER LARGE OBJECT value *agml.*

16) The null-call type-preserving SQL-invoked constructor method *ST_Direction(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_DirectionFrmGML(agml)*.

16.2.3 ST_Radians Method

Purpose

Observe the ST_Direction value as a DOUBLE PRECISION value in radians, representing clockwise rotation from True North.

Definition

```
CREATE METHOD ST_Radians()
   RETURNS DOUBLE PRECISION
   FOR ST_Direction
   RETURN SELF.ST_PrivateAngleNAzimuth.ST_Radians()
```

Description

1) The method *ST_Radians()* has no input parameters.
2) The null-call method *ST_Radians()* returns the value of the *ST_PrivateRadians* attribute of the value of the *ST_PrivateAngleNAzimuth* attribute of the *ST_Direction* value.

16.2.4 ST_AngleNAzimuth Methods

Purpose

Observe and mutate the North azimuth angle attribute of an ST_Direction value.

Definition

```
CREATE METHOD ST_AngleNAzimuth()
   RETURNS ST_Angle
   FOR ST_Direction
   RETURN SELF.ST_PrivateAngleNAzimuth

CREATE METHOD ST_AngleNAzimuth
   (nazimuthangle ST_Angle)
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      IF nazimuthangle IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         RETURN
            CASE
               WHEN SELF IS NULL THEN
                  NULL
               ELSE
                  SELF.ST_PrivateAngleNAzimuth(nazimuthangle)
            END;
      END IF;
   END
```

Description

1) The method *ST_AngleNAzimuth()* has no input parameters.

2) The null-call method *ST_AngleNAzimuth()* returns the value of the *ST_PrivateAngleNAzimuth* attribute.

3) The method *ST_AngleNAzimuth(ST_Angle)* takes the following input parameters:

   a) an *ST_Angle* value *nazimuthangle*.

4) For the type-preserving method *ST_AngleNAzimuth(ST_Angle)*:

   Case:

   a) If *nazimuthangle* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) If *nazimuthangle.ST_Radians()* is less than 0 (zero) radians or *nazimuthangle.ST_Radians()* is greater than or equal to (2π) radians , then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

   d) Otherwise, return the result of the value expression: *SELF.ST_PrivateAngleNAzimuth(nazimuthangle)*.

16.2.5 ST_AsText Method

Purpose

Return the well-known text representation of an ST_Direction value.

Definition

```
CREATE METHOD ST_AsText()
   RETURNS CHARACTER VARYING(ST_MaxDirectionAsText)
   FOR ST_Direction
   RETURN 'DIRECTION(N ' ||
      CAST (SELF.ST_PrivateAngleNAzimuth.ST_Radians()
         AS CHARACTER VARYING(ST_MaxDirectionAsText-13)) ||
      ')'
```

Definitional Rules

1) *ST_MaxDirectionAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Direction* value.

Description

1) The method *ST_AsText()* has no input parameters.
2) The null-call method *ST_AsText()* returns a CHARACTER VARYING value containing the well-known text representation of SELF. Values shall be produced in the BNF for <direction text representation>.

16.2.6 ST_GMLToSQL Method

Purpose

Return an ST_Direction value for a given GML representation.

Definition

```
CREATE METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML))
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.

Description

1) The method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The parameter *agml* is the GML Direction representation of an *ST_Direction* value. If *agml* does not contain a Direction XML element in the GML representation that can be transformed into an ST_Direction value, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
3) The null-call method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* returns an *ST_Direction* value represented by *agml*.

16.2.7 ST_AsGML Method

Purpose

Return the GML representation of an ST_Direction value.

Definition

```
CREATE METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML)
   FOR ST_Direction
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.

Description

1) The method *ST_AsGML()* has no input parameters.
2) The null-call method *ST_AsGML()* returns a CHARACTER LARGE OBJECT value containing a GML representation. The instantiable type of ST_Direction is mapped to a Direction XML element in the GML representation.

16.2.8 ST_RadianBearing Method

Purpose

Observe the ST_Direction value as a bearing with its angle part expressed in radians.

Definition

```
CREATE METHOD ST_RadianBearing
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE NAZ ST_Angle;
      DECLARE A ST_Angle;
      DECLARE NS CHARACTER(1);
      DECLARE EW CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-6)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET NAZ = SELF.ST_PrivateAngleNAzimuth;
      IF (NAZ.ST_Degrees >= 0) AND
         (NAZ.ST_Degrees <= 90) THEN
         SET NS = 'N';
         SET A = NAZ;
         SET EW = 'E';
      ELSEIF (NAZ.ST_Degrees > 90) AND
         (NAZ.ST_Degrees <= 180) THEN
         SET NS = 'S';
         SET A = NEW ST_Angle('D', 180).ST_Subtract(NAZ);
         SET EW = 'E';
      ELSEIF (NAZ.ST_Degrees > 180) AND
         (NAZ.ST_Degrees < 270) THEN
         SET NS = 'S';
         SET A = NAZ.ST_Subtract(NEW ST_Angle('D', 180));
         SET EW = 'W';
      ELSEIF (NAZ.ST_Degrees >= 270) AND
         (NAZ.ST_Degrees < 360) THEN
         SET NS = 'N';
         SET A = NEW ST_Angle('D', 360).ST_Subtract(NAZ);
         SET EW = 'W';
      END IF;
      RETURN NS || ' ' ||
         CAST (ROUND(A.ST_Radians(), numdecdigits)
            AS CHARACTER VARYING(ST_MaxDirectionString)) ||
         ' ' || EW;
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_RadianBearing(INTEGER)* takes the following input parameters:

   a) an INTEGER value *numdecdigits*.

2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 6, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) Let *NAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth*.

4) Case:

   a) If 0 <= *NAZ.ST_Degrees()* <= 90, then:

      i) Let *NS* be the CHARACTER value equal to 'N'.

      ii) Let *A* be the *ST_Angle* value equal to *NAZ*.

      iii) Let *EW* be the CHARACTER value equal to 'E'.

   b) If 90 < *NAZ.ST_Degrees()* <= 180, then:

      i) Let *NS* be the CHARACTER value equal to 'S'.

      ii) Let *A* be the *ST_Angle* value equal to *NEW ST_Angle*('D',180).*ST_Subtract*(*NAZ*).

      iii) Let *EW* be the CHARACTER value equal to 'E'.

   c) If 180 < *NAZ.ST_Degrees()* < 270, then:

      i) Let *NS* be the CHARACTER value equal to 'S'.

      ii) Let *A* be the *ST_Angle* value equal to *NAZ.ST_Subtract(NEW ST_Angle('D',180))*.

      iii) Let *EW* be the CHARACTER value equal to 'W'.

   d) If 270 <= *NAZ.ST_Degrees()* < 360, then:

      i) Let *NS* be the CHARACTER value equal to 'N'.

      ii) Let *A* be the *ST_Angle* value equal to *NEW ST_Angle('D',360).ST_Subtract(NAZ)*.

      iii) Let *EW* be the CHARACTER value equal to 'W'.

5) The null-call method *ST_RadianBearing(INTEGER)* returns the value of the *ST_Direction* as a bearing measured in radians and expressed as a character string concatenated from:

   a) the *NS* CHARACTER value,

   b) a space CHARACTER value,

   c) the *A.ST_Radians()* DOUBLE PRECISION value rounded or truncated to the number of decimal places indicated by *numdecdigits* and then expressed as a CHARACTER VARYING value. The choice of whether to truncate or round is implementation-defined.

   d) a space CHARACTER value,

   e) the *EW* CHARACTER value.

6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.9 ST_DegreesBearing Method

Purpose

Observe the ST_Direction value as a bearing with its angle part expressed in decimal degrees.

Definition

```
CREATE METHOD ST_DegreesBearing
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE NAZ ST_Angle;
      DECLARE A ST_Angle;
      DECLARE NS CHARACTER(1);
      DECLARE EW CHARACTER(1);
      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-7)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET NAZ = SELF.ST_PrivateAngleNAzimuth;
      IF (NAZ.ST_Degrees >= 0) AND
         (NAZ.ST_Degrees <= 90) THEN
         SET NS = 'N';
         SET A = NAZ;
         SET EW = 'E';
      ELSEIF (NAZ.ST_Degrees > 90) AND
         (NAZ.ST_Degrees <= 180) THEN
         SET NS = 'S';
         SET A = NEW ST_Angle('D', 180).ST_Subtract(NAZ);
         SET EW = 'E';
      ELSEIF (NAZ.ST_Degrees > 180) AND
         (NAZ.ST_Degrees < 270) THEN
         SET NS = 'S';
         SET A = NAZ.ST_Subtract(NEW ST_Angle('D', 180));
         SET EW = 'W';
      ELSEIF (NAZ.ST_Degrees >= 270) AND
         (NAZ.ST_Degrees < 360) THEN
         SET NS = 'N';
         SET A = NEW ST_Angle('D', 360).ST_Subtract(NAZ);
         SET EW = 'W';
      END IF;
      RETURN NS || ' ' ||
         CAST (ROUND(A.ST_Degrees(), numdecdigits)
            AS CHARACTER VARYING(ST_MaxDirectionString)) ||
         ' ' || EW;
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_DegreesBearing(INTEGER)* takes the following input parameters:

   a) an INTEGER value *numdecdigits*.

2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 7, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) Let *NAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth*.

4) Case:
   a) If 0 <= *NAZ.ST_Degrees()* <= 90, then:
      i) Let *NS* be the CHARACTER value equal to 'N'.
      ii) Let *A* be the *ST_Angle* value equal to *NAZ*.
      iii) Let *EW* be the CHARACTER value equal to 'E'.
   b) If 90 < *NAZ.ST_Degrees()* <= 180, then:
      i) Let *NS* be the CHARACTER value equal to 'S'.
      ii) Let *A* be the *ST_Angle* value equal to *NEW ST_Angle('D',180).ST_Subtract(NAZ)*.
      iii) Let *EW* be the CHARACTER value equal to 'E'.
   c) If 180 < *NAZ.ST_Degrees()* < 270, then:
      i) Let *NS* be the CHARACTER value equal to 'S'.
      ii) Let *A* be the *ST_Angle* value equal to *NAZ.ST_Subtract(NEW ST_Angle('D',180))*.
      iii) Let *EW* be the CHARACTER value equal to 'W'.
   d) If 270 <= *NAZ.ST_Degrees()* < 360, then:
      i) Let *NS* be the CHARACTER value equal to 'N'.
      ii) Let *A* be the *ST_Angle* value equal to *NEW ST_Angle('D',360).ST_Subtract(NAZ)*.
      iii) Let *EW* be the CHARACTER value equal to 'W'.
5) The null-call method *ST_DegreesBearing(INTEGER)* returns the value of the *ST_Direction* as a bearing measured in degrees and expressed as a character string concatenated from:
   a) the *NS* CHARACTER value,
   b) a space CHARACTER value,
   c) the *A.ST_Degrees()* DOUBLE PRECISION value rounded or truncated to the number of decimal places indicated by *numdecdigits* and then expressed as a CHARACTER VARYING value. The choice of whether to truncate or round is implementation-defined.
   d) a space CHARACTER value,
   e) the *EW* CHARACTER value.
6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.10 ST_DMSBearing Method

Purpose

Observe the ST_Direction value as a bearing with its angle part expressed in degrees, minutes, and seconds.

Definition

```
CREATE METHOD ST_DMSBearing
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE NAZ ST_Angle;
      DECLARE A ST_Angle;
      DECLARE NS CHARACTER(1);
      DECLARE EW CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-13)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET NAZ = SELF.ST_PrivateAngleNAzimuth;
      IF (NAZ.ST_Degrees >= 0) AND
         (NAZ.ST_Degrees <= 90) THEN
         SET NS = 'N';
         SET A = NAZ;
         SET EW = 'E';
      ELSEIF (NAZ.ST_Degrees > 90) AND
         (NAZ.ST_Degrees <= 180) THEN
         SET NS = 'S';
         SET A = NEW ST_Angle('D', 180).ST_Subtract(NAZ);
         SET EW = 'E';
      ELSEIF (NAZ.ST_Degrees > 180) AND
         (NAZ.ST_Degrees < 270) THEN
         SET NS = 'S';
         SET A = NAZ.ST_Subtract(NEW ST_Angle('D', 180));
         SET EW = 'W';
      ELSEIF (NAZ.ST_Degrees >= 270) AND
         (NAZ.ST_Degrees < 360) THEN
         SET NS = 'N';
         SET A = NEW ST_Angle('D', 360).ST_Subtract(NAZ);
         SET EW = 'W';
      END IF;
      RETURN NS || ' ' ||
         A.ST_String(numdecdigits) ||
         ' ' || EW;
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_DMSBearing(INTEGER)* takes the following input parameters:

   a) an INTEGER value *numdecdigits*.

2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 13, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) Let *NAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth*.

4) Case:

   a) If 0 <= *NAZ.ST_Degrees()* <= 90, then:

      i) Let *NS* be the CHARACTER value equal to 'N'.

      ii) Let *A* be the *ST_Angle* value equal to *NAZ*.

      iii) Let *EW* be the CHARACTER value equal to 'E'.

   b) If 90 < *NAZ.ST_Degrees()* <= 180, then:

      i) Let *NS* be the CHARACTER value equal to 'S'.

      ii) Let *A* be the *ST_Angle* value equal to *NEW ST_Angle('D',180).ST_Subtract(NAZ)*.

      iii) Let *EW* be the CHARACTER value equal to 'E'.

   c) If 180 < *NAZ.ST_Degrees()* < 270, then:

      i) Let *NS* be the CHARACTER value equal to 'S'.

      ii) Let *A* be the *ST_Angle* value equal to *NAZ.ST_Subtract(NEW ST_Angle('D',180))*.

      iii) Let *EW* be the CHARACTER value equal to 'W'.

   d) If 270 <= *NAZ.ST_Degrees()* < 360, then:

      i) Let *NS* be the CHARACTER value equal to 'N'.

      ii) Let *A* be the *ST_Angle* value equal to *NEW ST_Angle('D',360).ST_Subtract(NAZ)*.

      iii) Let *EW* be the CHARACTER value equal to 'W'.

5) The null-call method *ST_DMSBearing(INTEGER)* returns the value of the *ST_Direction* as a bearing measured in degrees, minutes, and seconds, and expressed as a character string concatenated from:

   a) the *NS* CHARACTER value,

   b) a space CHARACTER value,

   c) the *A.ST_String(numdecdigits)* CHARACTER VARYING value,

   d) a space CHARACTER value,

   e) the *EW* CHARACTER value.

6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.11 ST_RadianNAzimuth Method

Purpose

Observe the ST_Direction value as a North azimuth with its angle part expressed in radians.

Definition

```
CREATE METHOD ST_RadianNAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE NAZ ST_Angle;
      DECLARE NS CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-4)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET NAZ = SELF.ST_PrivateAngleNAzimuth;
      SET NS = 'N';
      RETURN NS || ' ' ||
         CAST (ROUND(NAZ.ST_Radians(), numdecdigits)
            AS CHARACTER VARYING(ST_MaxDirectionString));
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_RadianNAzimuth(INTEGER)* takes the following input parameters:
   a) an INTEGER value *numdecdigits*.
2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 4, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
3) Let *NAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth*.
4) Let *NS* be the CHARACTER value equal to 'N'.
5) The null-call method *ST_RadianNAzimuth(INTEGER)* returns the value of the *ST_Direction* as a North azimuth measured in radians and expressed as a character string concatenated from:
   a) the *NS* CHARACTER value,
   b) a space CHARACTER value,
   c) the *NAZ.ST_Radians()* DOUBLE PRECISION value rounded or truncated to the number of decimal places indicated by *numdecdigits* and then expressed as a CHARACTER VARYING value. The choice of whether to truncate or round is implementation-defined.
6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.12 ST_DegreesNAzimuth Method

Purpose

Observe the ST_Direction value as a North azimuth with its angle part expressed in decimal degrees.

Definition

```
CREATE METHOD ST_DegreesNAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING (ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE NAZ ST_Angle;
      DECLARE NS CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-6)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET NAZ = SELF.ST_PrivateAngleNAzimuth;
      SET NS = 'N';
      RETURN NS || ' ' ||
         CAST (ROUND(NAZ.ST_Degrees(), numdecdigits)
            AS CHARACTER VARYING(ST_MaxDirectionString));
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_DegreesNAzimuth(INTEGER)* takes the following input parameters:
   a) an INTEGER value *numdecdigits*.
2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 6, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
3) Let *NAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth*.
4) Let *NS* be the CHARACTER value equal to 'N'.
5) The null-call method *ST_DegreesNAzimuth(INTEGER)* returns the value of the *ST_Direction* as a North azimuth measured in degrees and expressed as a character string concatenated from:
   a) the *NS* CHARACTER value,
   b) a space CHARACTER value,
   c) the *NAZ.ST_Degrees()* DOUBLE PRECISION value rounded or truncated to the number of decimal places indicated by *numdecdigits* and then expressed as a CHARACTER VARYING value. The choice of whether to truncate or round is implementation-defined.
6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.13 ST_DMSNAzimuth Method

Purpose

Observe the ST_Direction value as a North azimuth with its angle part expressed in degrees, minutes, and seconds.

Definition

```
CREATE METHOD ST_DMSNAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING (ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE NAZ ST_Angle;
      DECLARE NS CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-12)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      SET NAZ = SELF.ST_PrivateAngleNAzimuth;
      SET NS = 'N';
      RETURN NS || ' ' ||
         NAZ.ST_String(numdecdigits);
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_DMSNAzimuth(INTEGER)* takes the following input parameters:
   a) an INTEGER value *numdecdigits*.
2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 12, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
3) Let *NAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth*.
4) Let *NS* be the CHARACTER value equal to 'N'.
5) The null-call method *ST_DMSNAzimuth(INTEGER)* returns the value of the *ST_Direction* as a North azimuth measured in degrees, minutes, and seconds, and expressed as a character string concatenated from:
   a) the *NS* CHARACTER value,
   b) a space CHARACTER value,
   c) the *NAZ.ST_String(numdecdigits)* CHARACTER VARYING value
6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.14 ST_RadianSAzimuth Method

Purpose

Observe the ST_Direction value as a South azimuth with its angle part expressed in radians.

Definition

```
CREATE METHOD ST_RadianSAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE SAZ ST_Angle;
      DECLARE NS CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-4)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      IF (SELF.ST_PrivateAngleNAzimuth.ST_Degrees() >= 0) AND
         (SELF.ST_PrivateAngleNAzimuth.ST_Degrees() < 180) THEN
         SET SAZ = SELF.ST_PrivateAngleNAzimuth.ST_Add(
            NEW ST_Angle('D',180));
      ELSE
         SET SAZ = SELF.ST_PrivateAngleNAzimuth.ST_Subtract(
            NEW ST_Angle('D',180));
      END IF;
      SET NS = 'S';
      RETURN NS || ' ' ||
         CAST (ROUND(SAZ.ST_Radians(), numdecdigits)
            AS CHARACTER VARYING(ST_MaxDirectionString));
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_RadianSAzimuth(INTEGER)* takes the following input parameters:

   a) an INTEGER value *numdecdigits*.

2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 4, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) Case:

   a) If 0 (zero) <= *SELF.ST_PrivateAngleNAzimuth.ST_Degrees()* < 180, then let *SAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth.ST_Add(NEW ST_Angle('D',180))*.

   b) Otherwise, let *SAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth.ST_Subtract(NEW ST_Angle('D',180))*.

4) Let *NS* be the CHARACTER value equal to 'S'.

5) The null-call method *ST_RadianSAzimuth(INTEGER)* returns the value of the *ST_Direction* as a South azimuth measured in radians and expressed as a character string concatenated from:

   a) the *NS* CHARACTER value,

   b) a space CHARACTER value,

c) the *SAZ.ST_Radians()* DOUBLE PRECISION value rounded or truncated to the number of decimal places indicated by *numdecdigits* and then expressed as a CHARACTER VARYING value. The choice of whether to truncate or round is implementation-defined.

6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.15 ST_DegreesSAzimuth Method

Purpose

Observe the ST_Direction value as a South azimuth with its angle part expressed in decimal degrees.

Definition

```
CREATE METHOD ST_DegreesSAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE SAZ ST_Angle;
      DECLARE NS CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-6)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      IF (SELF.ST_PrivateAngleNAzimuth.ST_Degrees() >= 0) AND
         (SELF.ST_PrivateAngleNAzimuth.ST_Degrees() < 180) THEN
         SET SAZ = SELF.ST_PrivateAngleNAzimuth.ST_Add(
            NEW ST_Angle('D',180));
      ELSE
         SET SAZ = SELF.ST_PrivateAngleNAzimuth.ST_Subtract(
            NEW ST_Angle('D',180));
      END IF;
      SET NS = 'S';
      RETURN NS || ' ' ||
         CAST (ROUND(SAZ.ST_Degrees(), numdecdigits)
            AS CHARACTER VARYING(ST_MaxDirectionString));
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_DegreesSAzimuth(INTEGER)* takes the following input parameters:

   a) an INTEGER value *numdecdigits*.

2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 6, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.

3) Case:

   a) If 0 (zero) <= *SELF.ST_PrivateAngleNAzimuth.ST_Degrees()* < 180, then let *SAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth.ST_Add(NEW ST_Angle('D',180))*.

   b) Otherwise, let *SAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth.ST_Subtract(NEW ST_Angle('D',180))*.

4) Let *NS* be the CHARACTER value equal to 'S'.

5) The null-call method *ST_DegreesSAzimuth(INTEGER)* returns the value of the *ST_Direction* as a South azimuth measured in degrees and expressed as a character string concatenated from:

   a) the *NS* CHARACTER value,

   b) a space CHARACTER value,

c) the *SAZ.ST_Degrees()* DOUBLE PRECISION value rounded or truncated to the number of decimal places indicated by *numdecdigits* and then expressed as a CHARACTER VARYING value. The choice of whether to truncate or round is implementation-defined.

6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.16 ST_DMSSAzimuth Method

Purpose

Observe the ST_Direction value as a South azimuth with its angle part expressed in degrees, minutes, and seconds.

Definition

```
CREATE METHOD ST_DMSSAzimuth
   (numdecdigits INTEGER)
   RETURNS CHARACTER VARYING(ST_MaxDirectionString)
   FOR ST_Direction
   BEGIN
      DECLARE SAZ ST_Angle;
      DECLARE NS CHARACTER(1);

      IF (numdecdigits < 0) OR
         (numdecdigits > (ST_MaxDirectionString-12)) THEN
         SIGNAL SQLSTATE '2FF02'
            SET MESSAGE_TEXT = 'invalid argument';
      END IF;
      IF (SELF.ST_PrivateAngleNAzimuth.ST_Degrees() >= 0) AND
         (SELF.ST_PrivateAngleNAzimuth.ST_Degrees() < 180) THEN
         SET SAZ = SELF.ST_PrivateAngleNAzimuth.ST_Add(
            NEW ST_Angle('D',180));
      ELSE
         SET SAZ = SELF.ST_PrivateAngleNAzimuth.ST_Subtract(
            NEW ST_Angle('D',180));
      END IF;
      SET NS = 'S';
      RETURN NS || ' ' ||
         SAZ.ST_String(numdecdigits);
   END
```

Definitional Rules

1) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

Description

1) The method *ST_DMSSAzimuth(INTEGER)* takes the following input parameters:
   a) an INTEGER value *numdecdigits*.
2) If *numdecdigits* is less than 0 (zero) or *numdecdigits* is greater than *ST_MaxDirectionString* minus 12, then an exception condition is raised: *SQL/MM Spatial exception – invalid argument*.
3) Case:
   a) If 0 (zero) <= *SELF.ST_PrivateAngleNAzimuth.ST_Degrees()* < 180, then let *SAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth.ST_Add(NEW ST_Angle('D',180))*.
   b) Otherwise, let *SAZ* be the *ST_Angle* value equal to *SELF.ST_PrivateAngleNAzimuth.ST_Subtract(NEW ST_Angle('D',180))*.
4) Let *NS* be the CHARACTER value equal to 'S'.
5) The null-call method *ST_DMSSAzimuth(INTEGER)* returns the value of the *ST_Direction* as a South azimuth measured in degrees, minutes, and seconds, and expressed as a character string concatenated from:
   a) the *NS* CHARACTER value,
   b) a space CHARACTER value,
   c) the *SAZ.ST_String(numdecdigits)* CHARACTER VARYING value.

6) The maximum measure value of *numdecdigits* is implementation-defined.

16.2.17 ST_AddAngle Method

Purpose

Mutate the ST_Direction value by adding an angle.

Definition

```
CREATE METHOD ST_AddAngle
   (anangle ST_Angle)
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      DECLARE resultant DOUBLE PRECISION;

      SET resultant =
         SELF.ST_PrivateAngleNAzimuth.ST_PrivateRadians +
            anangle.ST_PrivateRadians;
      IF resultant > (2*ST_ApproximatePi) THEN
         WHILE resultant > (2*ST_ApproximatePi) DO
            SET resultant = resultant - (2*ST_ApproximatePi);
         END WHILE;
      ELSEIF resultant < 0 THEN
         WHILE resultant < 0 DO
            SET resultant = resultant + (2*ST_ApproximatePi);
         END WHILE;
      END IF;
      RETURN SELF.ST_PrivateAngleNAzimuth(resultant);
   END
```

Definitional Rules

1) *ST_ApproximatePi* is the implementation-defined meta-variable representing π.

Description

1) The method *ST_AddAngle(ST_Angle)* takes the following input parameters:

   a) an *ST_Angle* value *anangle*.

2) The null-call type-preserving method *ST_AddAngle(ST_Angle)* returns the value of SELF with the *SELF.ST_PrivateAngleNAzimuth* attribute set to the angle value constructed from radians, with the number of radians being the sum of *SELF.ST_PrivateAngleNAzimuth.ST_PrivateRadians* plus *anangle.ST_PrivateRadians* modified as follows:

   Case:

   a) If the resultant sum is greater than or equal to 2π, then 2π is repeatedly subtracted until the result is less than 2π.

   b) If the resultant sum is less than 0 (zero), then 2π is repeatedly added until the result is greater than or equal to 0 (zero).

16.2.18 ST_SubtractAngle Method

Purpose

Mutate the ST_Direction value by subtracting an angle.

Definition

```
CREATE METHOD ST_SubtractAngle
   (anangle ST_Angle)
   RETURNS ST_Direction
   FOR ST_Direction
   BEGIN
      DECLARE resultant DOUBLE PRECISION;

      SET resultant =
         SELF.ST_PrivateAngleNAzimuth.ST_PrivateRadians -
            anangle.ST_PrivateRadians;
      IF resultant > (2*ST_ApproximatePi) THEN
         WHILE resultant > (2*ST_ApproximatePi) DO
            SET resultant = resultant - (2*ST_ApproximatePi);
         END WHILE;
      ELSEIF resultant < 0 THEN
         WHILE resultant < 0 DO
            SET resultant = resultant + (2*ST_ApproximatePi);
         END WHILE;
      END IF;
      RETURN SELF.ST_PrivateAngleNAzimuth(resultant);
   END
```

Definitional Rules

1) *ST_ApproximatePi* is the implementation-defined meta-variable representing π.

Description

1) The method *ST_SubtractAngle(ST_Angle)* takes the following input parameters:

   a) an *ST_Angle* value *anangle*.

2) The null-call type-preserving method *ST_SubtractAngle(ST_Angle)* returns the value of SELF with the *SELF.ST_PrivateAngleNAzimuth* attribute set to the angle value constructed from radians, with the number of radians being the difference of *SELF.ST_PrivateAngleNAzimuth.ST_PrivateRadians* minus *anangle.ST_PrivateRadians*, modified as follows:

   Case:

   a) If the resultant difference is greater than or equal to 2π, then 2π is repeatedly subtracted until the result is less than 2π.

   b) If the resultant difference is less than 0 (zero), then 2π is repeatedly added until the result is greater than or equal to 0 (zero).

16.2.19 ST_DirectionFrmTxt Function

Purpose

Return an ST_Direction value which is transformed from a CHARACTER VARYING value that represents the well-known text representation of an ST_Direction value.

Definition

```
CREATE FUNCTION ST_DirectionFrmTxt
   (awkt CHARACTER VARYING(ST_MaxDirectionAsText))
   RETURNS ST_Direction
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxDirectionAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Direction* value.

Description

1) The function *ST_DirectionFrmTxt(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *awkt*.
2) For the null-call function *ST_DirectionFrmTxt(CHARACTER VARYING)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Direction* value.

      If *awkt* is not producible in the BNF for <direction text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

   b) Otherwise, return the result of the value expression: *TREAT(ST_DirectionFrmTxt(awkt) AS ST_Direction)*.

16.2.20 ST_DirectionFrmGML Function

Purpose

Return an ST_Direction value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Direction representation of an ST_Direction value.

Definition

```
CREATE FUNCTION ST_DirectionFrmGML
   (agml CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML))
   RETURNS ST_Direction
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.

Description

1) The function *ST_DirectionFrmGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) For the null-call function *ST_DirectionFrmGML(CHARACTER LARGE OBJECT, INTEGER)*:
   a) If the parameter *agml* does not contain a Direction XML element in the GML representation that can be transformed into an ST_Direction value, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Otherwise, return the result of the value expression: *TREAT(ST_DirectionFrmGML(agml) AS ST_Direction)*.

16.2.21 ST_Direction Ordering Definition

Purpose

Define ordering for ST_Direction.

Definition

```
CREATE FUNCTION ST_OrderingCompare
   (adirection ST_Direction,
    anotherdirection ST_Direction)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN
      CASE
         WHEN adirection.ST_PrivateAngleNAzimuth <
               anotherdirection.ST_PrivateAngleNAzimuth THEN
            -1
         WHEN adirection.ST_PrivateAngleNAzimuth >
               anotherdirection.ST_PrivateAngleNAzimuth THEN
            1
         ELSE
            0
      END

CREATE ORDERING FOR ST_Direction
   ORDER FULL BY RELATIVE
   WITH FUNCTION ST_OrderingCompare(ST_Direction, ST_Direction)
```

Description

1) The function *ST_OrderingCompare(ST_Direction, ST_Direction)* takes the following input parameters:
   a) an *ST_Direction* value *adirection*,
   b) an *ST_Direction* value *anotherdirection*.
2) For the null-call function *ST_OrderingCompare(ST_Direction, ST_Direction)*:

   Case:
   a) If *adirection.ST_PrivateAngleNAzimuth* < *anotherdirection.ST_PrivateAngleNAzimuth*, then return -1.
   b) If *adirection.ST_PrivateAngleNAzimuth* > *anotherdirection.ST_PrivateAngleNAzimuth*, then return 1 (one).
   c) Otherwise, return 0 (zero).
3) Use the function *ST_OrderingCompare(ST_Direction, ST_Direction)* to define ordering for the *ST_Direction* type.

16.2.22 SQL Transform Functions

Purpose

Define SQL transform functions for the ST_Direction type.

Definition

```
CREATE TRANSFORM FOR ST_Direction
   ST_WellKnownText
      (TO SQL WITH METHOD ST_Direction
         (CHARACTER VARYING(ST_MaxDirectionAsText)),
       FROM SQL WITH METHOD ST_AsText())
   ST_WellKnownBinary
      (TO SQL WITH METHOD ST_Direction(DOUBLE PRECISION),
       FROM SQL WITH METHOD ST_Radians())
   ST_GML
      (TO SQL WITH METHOD ST_GMLToSQL
         (CHARACTER LARGE OBJECT(ST_MaxDirectionAsGML)),
       FROM SQL WITH METHOD ST_AsGML())
```

Definitional Rules

1) *ST_MaxDirectionAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Direction* value.

2) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.

Description

1) Use the method *ST_Direction(CHARACTER VARYING)* and the method *ST_AsText()* to define the transform group *ST_WellKnownText*.

2) Use the method *ST_Direction(DOUBLE PRECISION)* and the method *ST_Radians()* to define the transform group *ST_WellKnownBinary*.

3) Use the method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsGML()* to define the transform group *ST_GML*.

16.2.23 <direction text representation>

Purpose

This subclause contains the definition of the <well-known text representation> of an ST_Direction value.

Description

1) The well-known text representation of an *ST_Direction* value is defined by the following BNF for <direction text representation>:

```
<direction text representation> ::=
     DIRECTION <nazimuth text>

<nazimuth text> ::=
     <left paren> N <radians> <right paren>

<radians> ::=
     <number>

<left paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<number> ::=
     <exact numeric literal>
   | <approximate numeric literal>

<exact numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<approximate numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075
```

a) <direction text representation> is the well-known text representation for an *ST_Direction* value that is produced by <nazimuth text>.

b) <nazimuth text> produces the *ST_Direction* value from <radians>.

c) Let *RADIANS* be the DOUBLE PRECISION value specified by <radians>. Then <radians> produces an *ST_Direction* value as the result of the value expression: *NEW ST_Direction('N', NEW ST_Angle('R', RADIANS))*.

d) The list of keywords is: DIRECTION and N.

17 Support Types

17.1 ST_TINElement Type and Routines

17.1.1 ST_TINElement Type

Purpose

The ST_TINElement type is used to specify the information used to construct an ST_TIN surface. Element types include random points, group spot, boundary, breakline, soft break, control contour, break void, drape void, void, hole, stop line and user defined element types.

Definition

```
CREATE TYPE ST_TINElement
   AS (
      ST_PrivateElementType CHARACTER VARYING(30) DEFAULT NULL,
      ST_PrivateElementID INTEGER DEFAULT NULL,
      ST_PrivateElementTag CHARACTER VARYING(64) DEFAULT NULL,
      ST_PrivateElementGeometry ST_Geometry DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_TINElement
      (elementtype CHARACTER VARYING(30),
       elementID INTEGER,
       elementtag CHARACTER VARYING(64),
       elementgeometry ST_Geometry)
      RETURNS ST_TINElement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_ElementType()
      RETURNS CHARACTER VARYING(30)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_ElementType
      (elementtype CHARACTER VARYING(30))
      RETURNS ST_TINElement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_ElementID()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_ElementID
   (elementID INTEGER)
   RETURNS ST_TINElement
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ElementTag()
   RETURNS CHARACTER VARYING(64)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ElementTag
   (elementtag CHARACTER VARYING(64))
   RETURNS ST_TINElement
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_ElementGeometry()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_ElementGeometry
   (elementgeometry ST_Geometry)
   RETURNS ST_TINElement
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateElementType* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateElementType*.

2) The attribute *ST_PrivateElementID* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateElementID*.

3) The attribute *ST_PrivateElementTag* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateElementTag*.

4) The attribute *ST_PrivateElementGeometry* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateElementGeometry*.

Description

1) The *ST_TINElement* type provides for public use:

   a) a method *ST_TINElement(CHARACTER VARYING, INTEGER, CHARACTER VARYING, ST_Geometry),*

   b) a method *ST_ElementType()*,

   c) a method *ST_ElementType(CHARACTER VARYING)*,

   d) a method *ST_ElementID()*,

   e) a method *ST_ElementID(INTEGER)*,

   f) a method *ST_ElementTag()*,

   g) a method *ST_ElementTag(CHARACTER VARYING)*,

   h) a method *ST_ElementGeometry()*,

   i) a method *ST_ElementGeometry(ST_Geometry)*,

   j) a method *ST_IsEmpty*().

2) An *ST_TINElement* value returned by the constructor function corresponds to the empty set.

3) The *ST_PrivateElementType* attribute is a CHARACTER VARYING value representing the type of TIN element.

4) Predefined TIN element types include: 'random points', 'group spot', 'boundary', 'breakline', 'soft break', 'control contour', 'break void', 'drape void', 'void', 'hole' and 'stop line'.

5) It is implementation-defined which of the predefined TIN element types are supported.

6) It is implementation-defined which additional TIN element types are supported, what type of *ST_Geometry* each requires, and what behavior is to be expected during triangulation.

7) The *ST_PrivateElementID* attribute is an INTEGER value representing a user assignable identifier for the TIN element.

8) The *ST_PrivateElementTag* attribute is a CHARACTER VARYING value representing a user assignable tag for the TIN element.

9) The *ST_PrivateElementGeometry* attribute is an *ST_Geometry* value representing the geometry of the TIN element.

10) A TIN element having a 'random points' element type represents points on the surface of known elevation from which triangles can be generated.

11) A TIN element having a 'group spot' element type represents a collection of related points on the surface of known elevation from which triangles can be generated. A 'group spot' type of TIN element will usually have a non-NULL *ST_PrivateElementID* or *ST_PrivateElementTag* to differentiate it from other 'group spot' TIN elements.

12) The TIN element having a 'boundary' element type can be used to define the boundary in the resulting *ST_TIN* value. When supplied to the *ST_TINElements* mutator method of the *ST_TIN*, the TIN surface value is clipped to the *ST_TINElement ST_Polygon* value. There can be at most only one such element for each *ST_TIN* value. It is implementation-defined whether interior boundaries are supported.

13) A TIN element having a 'breakline' element type represents a local ridge or depression in the TIN surface. When a breakline is specified for an *ST_TIN* value, triangles in the *ST_PrivatePatches* attribute of the *ST_TIN* value must be adjusted so that no triangle is crossed by the breakline. Part or all of the breakline becomes an edge of two or more triangles. The elevation along the breakline takes precedence over the elevation of the original TIN surface for the entire length of the breakline.

14) During (re)triangulation, *ST_Triangle* values in the vicinity of each breakline are adjusted so that, if the *ST_Triangle* value intersects the *ST_LineString* breakline value, then either:

a) one of the *ST_Point* values in the *ST_PrivateExteriorRing* attribute value of the *ST_Triangle* value is spatially 3D equal to one of the the *ST_Point* values in the *ST_PrivatePoints* attribute of the *ST_LineString* value of the breakline, or

b) two of the *ST_Point* values in the *ST_PrivateExteriorRing* attribute of the *ST_Triangle* value are spatially 3D equal to two consecutive *ST_Point* values in the *ST_PrivatePoints* attribute of the *ST_LineString* value of the breakline; the included side of the triangle lies along the breakline.

15) A TIN element having a 'soft break' element type behaves as a breakline (see 13, 14 above) except that countour lines generated for the surface can be smoothed where they cross soft breaks.

16) A TIN element having a 'control contour' element type behaves as a breakline (see 13, 14 above).

17) A TIN element having a 'break void', 'drape void' or 'void' element type encloses a voided area of the TIN surface.

18) When a break void is specified for an *ST_TIN* value, the boundary *ST_LineStrings* of the *ST_Polygon* behave as breaklines in that triangles in the *ST_PrivatePatches* attribute of the *ST_TIN* value must be adjusted so that no triangle is crossed by the break void boundary. Part or all of the break void boundary becomes an edge of two or more triangles. The elevation of this break void boundary takes precedence over the elevation of the original TIN surface for the entire length of the boundary.

19) When a drape void is specified for an *ST_TIN* value, the boundary *ST_LineStrings* of the *ST_Polygon* behave as breaklines in that triangles in the *ST_PrivatePatches* attribute of the *ST_TIN* value must be adjusted so that no triangle is crossed by the drape void boundary. Part or all of the drape void boundary becomes an edge of two or more triangles. However, for drape voids, the elevation of the original TIN surface takes precedence over the elevation of the drape void boundary.

20) When a (regular) void is specified for an *ST_TIN* value, the boundary *ST_LineStrings* of the *ST_Polygon* behave as breaklines in that triangles in the *ST_PrivatePatches* attribute of the *ST_TIN* value must be adjusted so that no triangle is crossed by the void boundary. Part or all of the void boundary becomes an edge of two or more triangles. However, for regular voids, only the elevations of the void boundary vertices take precedence over the elevation of corresponding points on the original surface. That is, these vertices are treated as points for triangulating. The regular void boundaries between these vertices are handled as drape void boundaries - elevations from the original surface take precedence.

21) A TIN element having a 'hole' element type encloses an area of the TIN surface designated as a hole.

22) When a hole is specified for an *ST_TIN* value, the boundary *ST_LineStrings* of the *ST_Polygon* behave as breaklines in that triangles in the *ST_PrivatePatches* attribute of the *ST_TIN* value must be adjusted so that no triangle is crossed by the hole boundary. Part or all of the hole boundary becomes an edge of two or more triangles. Hole boundaries are treated like drape void boundaries in that the elevation of the original TIN surface takes precedence over the elevation of the hole boundary.

23) The area bounded by TIN elements having a 'break void', 'drape void', 'void' and 'hole' element type are still considered to be part of the TIN surface, rather than a hole bounded by an interior boundary of the surface. This distinction is significant when merging two TIN surfaces.

24) TIN elements having a 'stop line' element type represent areas where the local continuity or regularity of the TIN surface is questionable.

25) It is implementation-defined whether the *ST_Triangle* values in the *ST_PrivatePatches* attribute whose boundaries are crossed by a stop line are removed from the *ST_PrivatePatches* attribute collection of *ST_Triangle* values. If they remain in the collection, it is implementation-defined whether they are enclosed within a 'drape void' type of *ST_TinElement*.

17.1.2 ST_TINElement Methods

Purpose

Return an ST_TINElement value constructed from the specified element type, element ID, element tag and element geometry values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_TINElement
   (elementtype CHARACTER VARYING(30),
    elementID INTEGER,
    elementtag CHARACTER VARYING(64),
    elementgeometry ST_Geometry)
   RETURNS ST_TINElement
   FOR ST_TINElement
   BEGIN
      DECLARE acounter INTEGER;
      DECLARE zee DOUBLE PRECISION;
      IF (((elementtype = 'random points') OR
           (elementtype = 'group spot')) AND
           (elementgeometry.ST_GeometryType() <> 'ST_MultiPoint')) OR
          (((elementtype = 'boundary') OR
            (elementtype = 'break void') OR
            (elementtype = 'drape void') OR
            (elementtype = 'void') OR
            (elementtype = 'hole')) AND
           (elementgeometry.ST_GeometryType() <> 'ST_Polygon')) OR
          (((elementtype = 'breakline') OR
            (elementtype = 'soft break') OR
            (elementtype = 'control contour') OR
            (elementtype = 'stop line')) AND
           (elementgeometry.ST_GeometryType() <> 'ST_LineString')) THEN
         SIGNAL SQLSTATE '2FF74'
         SET MESSAGE_TEXT = 'invalid geometry';
      ELSE
         IF (((elementtype = 'random points') OR
              (elementtype = 'group spot') OR
              (elementtype = 'boundary') OR
              (elementtype = 'breakline') OR
              (elementtype = 'soft break') OR
              (elementtype = 'control contour') OR
              (elementtype = 'break void') OR
              (elementtype = 'void')) AND
             (elementgeometry.ST_Is3D() = 0)) THEN
            SIGNAL SQLSTATE '2FF74'
            SET MESSAGE_TEXT = 'invalid geometry';
         ELSE
            IF (elementtype = 'control contour') THEN
               SET zee = TREAT(elementgeometry AS
               ST_LineString).ST_PointN(1).ST_Z();
               SET acounter = 2;
               WHILE acounter <= TREAT(elementgeometry AS
               ST_LineString).ST_NumPoints() DO
                  IF TREAT(elementgeometry AS
                  ST_LineString).ST_PointN(acounter).ST_Z() <> zee
                  THEN
                     SIGNAL SQLSTATE '2FF74'
                     SET MESSAGE_TEXT = 'invalid geometry';
                  END IF;
                  SET acounter = acounter + 1;
               END WHILE;
```

```
            ELSE
               RETURN SELF.ST_ElementType(elementtype).
                  ST_ElementID(elementID).
                  ST_ElementTag(elementtag).
                  ST_ElementGeometry(elementgeometry)
            END IF;
         END IF;
      END IF;
   END
```

Description

1) The method *ST_TINElement(CHARACTER VARYING(30), INTEGER, CHARACTER VARYING(64), ST_Geometry)* takes the following input parameters:

   a) a CHARACTER VARYING value *elementtype*,

   b) an INTEGER value *elementID*,

   c) a CHARACTER VARYING value *elementtag*,

   d) an *ST_Geometry* value *elementgeometry*.

2) For the null-call type-preserving SQL-invoked constructor method *ST_TINElement(CHARACTER VARYING(30), INTEGER, CHARACTER VARYING(64), ST_Geometry)*:

   Case:

   a) an exception condition is raised: *SQL/MM Spatial exception – invalid geometry* if any of the following conditions are not satisfied:

      i) if *elementtype* = 'random points', then *elementgeometry.ST_GeometryType()* = 'ST_MultiPoint' and *elementgeometry.ST_Is3D()* = 1 (one).

      ii) if *elementtype* = 'group spot', then *elementgeometry.ST_GeometryType()* = 'ST_MultiPoint' and *elementgeometry.ST_Is3D()* = 1 (one).

      iii) if *elementtype* = 'boundary' then *elementgeometry.ST_GeometryType()* = 'ST_Polygon' and *elementgeometry.ST_Is3D()* = 1 (one).

      iv) if *elementtype* = 'breakline', then *elementgeometry.ST_GeometryType()* = 'ST_LineString' and elementgeometry.ST_Is3D() = 1 (one).

      v) if *elementtype* = 'soft break', then *elementgeometry.ST_GeometryType()* = 'ST_LineString' and elementgeometry.ST_Is3D() = 1 (one).

      vi) if *elementtype* = 'control contour', then *elementgeometry.ST_GeometryType()* = 'ST_LineString' and *elementgeometry.ST_Is3D()* = 1 (one) and, for all of the ST_Point values in the ST_LineString value, ST_Z() returns the same value.

      vii) if *elementtype* = 'break void', then *elementgeometry.ST_GeometryType()* = 'ST_Polygon' and *elementgeometry.ST_Is3D()* = 1 (one).

      viii) if *elementtype* = 'drape void' then *elementgeometry.ST_GeometryType()* = 'ST_Polygon'.

      ix) if *elementtype* = 'void', then *elementgeometry.ST_GeometryType()* = 'ST_Polygon' and *elementgeometry.ST_Is3D()* = 1 (one).

      x) if *elementtype* = 'hole', then *elementgeometry.ST_GeometryType()* = 'ST_Polygon'.

      xi) if *elementtype* = 'stop line', then *elementgeometry.ST_GeometryType()* = 'ST_LineString'.

   b) otherwise, returns an ST_TINElement value with:

      i) Using the method *ST_ElementType(CHARACTER VARYING)*, the *ST_PrivateElementType* set to *elementtype*.

      ii) Using the method *ST_ElementID(INTEGER)*, the *ST_PrivateElementID* set to *elementID*.

      iii) Using the method *ST_ElementTag(CHARACTER VARYING)*, the *ST_PrivateElementTag* set to *elementtag*.

iv) Using the method *ST_ElementGeometry(ST_Geometry)*, the *ST_PrivateElementGeometry* set to *elementgeometry*.

17.1.3 ST_ElementType Methods

Purpose

Observe and mutate the ST_PrivateElementType attribute of the ST_TINElement value.

Definition

```
CREATE METHOD ST_ElementType()
   RETURNS CHARACTER VARYING(30)
   FOR ST_TINElement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateElementType
      END

CREATE METHOD ST_ElementType
   (elementtype CHARACTER VARYING(30))
   RETURNS ST_TINElement
   FOR ST_TINElement
   BEGIN
      -- If elementtype is the null value, then raise an exception
      IF elementtype IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               SELF.ST_PrivateElementType(elementtype)
         END;
   END
```

Description

1) The method *ST_ElementType()* has no input parameters.

2) For the null-call method *ST_ElementType()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateElementType* attribute of SELF.

3) The method *ST_ElementType(*CHARACTER VARYING*)* takes the following input parameters:

   a) a *CHARACTER VARYING* value *elementtype*.

4) For the type-preserving method *ST_ElementType(*CHARACTER VARYING*)*:

   Case:

   a) If *elementtype* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateElementType*(*elementtype)*.

17.1.4 ST_ElementID Methods

Purpose

Observe and mutate the ST_PrivateElementID attribute of the ST_TINElement value.

Definition

```
CREATE METHOD ST_ElementID()
   RETURNS INTEGER
   FOR ST_TINElement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateElementID
      END

CREATE METHOD ST_ElementID
   (elementID INTEGER)
   RETURNS ST_TINElement
   FOR ST_TINElement
   BEGIN
      -- If elementID is the null value, then raise an exception
      IF elementID IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               SELF.ST_PrivateElementID(elementID)
         END;
   END
```

Description

1) The method *ST_ElementID()* has no input parameters.
2) For the null-call method *ST_ElementID()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateElementID* attribute of SELF.
3) The method *ST_ElementID(*INTEGER*)* takes the following input parameters:
   a) an INTEGER value *ElementID*.
4) For the type-preserving method *ST_ElementID(*INTEGER*)*:

   Case:

   a) If *elementID* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateElementID*(*elementID)*.

17.1.5 ST_ElementTag Methods

Purpose

Observe and mutate the ST_PrivateElementTag attribute of the ST_TINElement value.

Definition

```
CREATE METHOD ST_ElementTag()
   RETURNS CHARACTER VARYING(64)
   FOR ST_TINElement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateElementTag
      END

CREATE METHOD ST_ElementTag
   (elementtag CHARACTER VARYING(64))
   RETURNS ST_TINElement
   FOR ST_TINElement
   BEGIN
      -- If elementtag is the null value, then raise an exception
      IF elementtag IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      RETURN
         CASE
            WHEN SELF IS NULL THEN
               NULL
            ELSE
               SELF.ST_PrivateElementTag(elementtag)
         END;
   END
```

Description

1) The method *ST_ElementTag()* has no input parameters.
2) For the null-call method *ST_ElementTag()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateElementTag* attribute of SELF.
3) The method *ST_ElementTag(*CHARACTER VARYING*)* takes the following input parameters:
   a) a CHARACTER VARYING value *elementtag*.
4) For the type-preserving method *ST_ElementTag(*CHARACTER VARYING*)*:

   Case:

   a) If *elementtag* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateElementTag*(*elementtag)*.

17.1.6 ST_ElementGeometry Methods

Purpose

Observe and mutate the ST_PrivateElementGeometry attribute of the ST_TINElement value.

Definition

```
CREATE METHOD ST_ElementGeometry()
   RETURNS ST_Geometry
   FOR ST_TINElement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateElementGeometry
      END

CREATE METHOD ST_ElementGeometry
   (elementgeometry ST_Geometry)
   RETURNS ST_TINElement
   FOR ST_TINElement
   BEGIN
      -- If elementgeometry is the null value, then raise an exception
      IF elementgeometry IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      IF SELF IS NULL THEN
         RETURN NULL;
      ELSE
         DECLARE acounter INTEGER;
         DECLARE zee DOUBLE PRECISION;
         IF (((SELF.ST_PrivateElementType = 'random points') OR
              (SELF.ST_PrivateElementType = 'group spot')) AND
              (elementgeometry.ST_GeometryType() <> 'ST_MultiPoint')) OR
             (((SELF.ST_PrivateElementType = 'boundary') OR
               (SELF.ST_PrivateElementType = 'break void') OR
               (SELF.ST_PrivateElementType = 'drape void') OR
               (SELF.ST_PrivateElementType = 'void') OR
               (SELF.ST_PrivateElementType = 'hole')) AND
              (elementgeometry.ST_GeometryType() <> 'ST_Polygon')) OR
             (((SELF.ST_PrivateElementType = 'breakline') OR
               (SELF.ST_PrivateElementType = 'soft break') OR
               (SELF.ST_PrivateElementType = 'control contour') OR
               (SELF.ST_PrivateElementType = 'stop line')) AND
              (elementgeometry.ST_GeometryType() <> 'ST_LineString')) THEN
            SIGNAL SQLSTATE '2FF74'
            SET MESSAGE_TEXT = 'invalid geometry';
         ELSE
            IF (((SELF.ST_PrivateElementType = 'random points') OR
                 (SELF.ST_PrivateElementType = 'group spot') OR
                 (SELF.ST_PrivateElementType = 'boundary') OR
                 (SELF.ST_PrivateElementType = 'breakline') OR
                 (SELF.ST_PrivateElementType = 'soft break') OR
                 (SELF.ST_PrivateElementType = 'control contour') OR
                 (SELF.ST_PrivateElementType = 'break void') OR
                 (SELF.ST_PrivateElementType = 'void')) AND
                (elementgeometry.ST_Is3D() = 0)) THEN
               SIGNAL SQLSTATE '2FF74'
               SET MESSAGE_TEXT = 'invalid geometry';
```

```
            ELSE
               IF (SELF.ST_PrivateElementType = 'control contour') THEN
                  SET zee = TREAT(elementgeometry AS
                  ST_LineString).ST_PointN(1).ST_Z();
                  SET acounter = 2;
                  WHILE acounter <= TREAT(elementgeometry AS
                  ST_LineString).ST_NumPoints(() DO
                     IF TREAT(elementgeometry AS
                     ST_LineString).ST_PointN(acounter).ST_Z() <> zee
                        THEN
                        SIGNAL SQLSTATE '2FF74'
                        SET MESSAGE_TEXT = 'invalid geometry';
                     END IF;
                     SET acounter = acounter + 1;
                  END WHILE;
               ELSE
                  RETURN SELF.ST_PrivateElementGeometry(elementgeometry);
               END IF;
            END IF;
         END IF;
      END IF;
   END
```

Description

1) The method *ST_ElementGeometry()* has no input parameters.

2) For the null-call method *ST_ElementGeometry()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateElementGeometry* attribute of SELF.

3) The method *ST_ElementGeometry(ST_Geometry)* takes the following input parameters:

   a) an *ST_Geometry* value e*lementgeometry*.

4) For the type-preserving method *ST_ElementGeometry(ST_Geometry)*:

   Case:

   a) If *elementgeometry* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) an exception condition is raised: *SQL/MM Spatial exception – invalid geometry* if any of the following conditions are not satisfied:

      i) if *SELF.ST_PrivateElementType* = 'random points', then *elementgeometry.ST_GeometryType()* = 'ST_MultiPoint' and *elementgeometry.ST_Is3D()* = 1 (one).

      ii) if *SELF.ST_PrivateElementType* = 'group spot', then *elementgeometry.ST_GeometryType()* = 'ST_MultiPoint' and *elementgeometry.ST_Is3D()* = 1 (one).

      iii) if *SELF.ST_PrivateElementType* = 'boundary' then *elementgeometry.ST_GeometryType()* = 'ST_Polygon' and *elementgeometry.ST_Is3D()* = 1 (one).

      iv) if *SELF.ST_PrivateElementType* = 'breakline', then *elementgeometry.ST_GeometryType()* = 'ST_LineString' and *elementgeometry.ST_Is3D()* = 1 (one).

      v) if *SELF.ST_PrivateElementType* = 'soft break', then *elementgeometry.ST_GeometryType()* = 'ST_LineString' and *elementgeometry.ST_Is3D()* = 1 (one).

vi) if *SELF.ST_PrivateElementType* = 'control contour', then *elementgeometry.ST_GeometryType()* = 'ST_LineString' and *elementgeometry.ST_Is3D()* = 1 (one) and, for all of the *ST_Point* values in the *ST_LineString* value, *ST_Z*() returns the same value.

vii) if *SELF.ST_PrivateElementType* = 'break void', then *elementgeometry.ST_GeometryType()* = 'ST_Polygon' and *elementgeometry.ST_Is3D()* = 1 (one).

viii) if *SELF.ST_PrivateElementType* = 'drape void' then *elementgeometry.ST_GeometryType()* = 'ST_Polygon'.

ix) if *SELF.ST_PrivateElementType* = 'void', then *elementgeometry.ST_GeometryType()* = 'ST_Polygon' and *elementgeometry.ST_Is3D()* = 1 (one).

x) if *SELF.ST_PrivateElementType* = 'hole', then *elementgeometry.ST_GeometryType()* = 'ST_Polygon'.

xi) if *SELF.ST_PrivateElementType* = 'stop line', then *elementgeometry.ST_GeometryType()* = 'ST_LineString'.

c) Otherwise, return the result of the value expression: *SELF.ST_PrivateElementGeometry*(*elementgeometry)*.

17.1.7 ST_IsEmpty Method

Test if an ST_TINElement value corresponds to the empty set.

Definition

```
CREATE METHOD ST_IsEmpty()
   RETURNS INTEGER
   FOR ST_TINElement
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsEmpty()* has no input parameters.
2) For the null-call method *ST_IsEmpty()*:

   Case:

   a) If the *ST_TINElement* value corresponds to the empty set, then return 1 (one).
   b) Otherwise, return 0 (zero).
3) An *ST_TINElement* value returned by the constructor function corresponds to the empty set.

17.2 ST_Vector Type and Routines

17.2.1 ST_Vector Type

Purpose

The ST_Vector type is an ordered set of numbers called coordinates that represent a magnitude and direction or a position in a coordinate system.

Definition

```
CREATE TYPE ST_Vector
   AS (
      ST_PrivateX DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateY DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateZ DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Vector
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxVectorAsText))
      RETURNS ST_Vector
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Vector
      (awktorgml CHARACTER LARGE OBJECT(ST_MaxVectorAsText),
       ansrid INTEGER)
      RETURNS ST_Vector
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Vector
      (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary))
      RETURNS ST_Vector
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   CONSTRUCTOR METHOD ST_Vector
      (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary),
       ansrid INTEGER)
      RETURNS ST_Vector
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_X()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_X
   (xcoord DOUBLE PRECISION)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,
```

```
METHOD ST_Y()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Y
   (ycoord DOUBLE PRECISION)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Z()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Z
   (zcoord DOUBLE PRECISION)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD ST_Coordinates()
   RETURNS DOUBLE PRECISION ARRAY[3]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Is3D()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SRID()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_SRID
   (ansrid INTEGER)
   RETURNS ST_Vector
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Equals
   (avector ST_Vector)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_WKTToSQL
   (awkt CHARACTER LARGE OBJECT(ST_MaxVectorAsText))
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsText()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxVectorAsText)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_WKBToSQL
   (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary))
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsBinary()
   RETURNS BINARY LARGE OBJECT(ST_MaxVectorAsBinary)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxVectorAsGML))
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxVectorAsGML)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.
2) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Vector* value.
3) *ST_MaxVectorAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Vector* value.
4) The attribute *ST_PrivateX* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateX*.
5) The attribute *ST_PrivateY* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateY*.
6) The attribute *ST_PrivateZ* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateZ*.

Description

1) The *ST_Vector* type provides for public use:
   a) a method *ST_Vector(CHARACTER LARGE OBJECT)*,
   b) a method *ST_Vector(CHARACTER LARGE OBJECT, INTEGER)*,
   c) a method *ST_Vector(BINARY LARGE OBJECT)*,
   d) a method *ST_Vector(BINARY LARGE OBJECT, INTEGER)*,
   e) a method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION)*,
   f) a method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,
   g) a method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)*,
   h) a method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*,
   i) a method *ST_X()*,
   j) a method *ST_X(DOUBLE PRECISION)*,
   k) a method *ST_Y()*,
   l) a method *ST_Y(DOUBLE PRECISION)*,
   m) a method *ST_Z()*,
   n) a method *ST_Z(DOUBLE PRECISION)*,
   o) a method *ST_Coordinates()*,
   p) a method *ST_Is3D()*,
   q) a method *ST_SRID()*,
   r) a method *ST_SRID(INTEGER)*,
   s) a method *ST_Equals(ST_Vector)*,
   t) a method *ST_IsEmpty()*,
   u) a method *ST_WKTToSQL(CHARACTER LARGE OBJECT)*,
   v) a method *ST_AsText()*,
   w) a method *ST_WKBToSQL(BINARY LARGE OBJECT)*,
   x) a method *ST_AsBinary()*,
   y) a method *ST_GMLToSQL(CHARACTER LARGE OBJECT)*,
   z) a method *ST_AsGML()*,

aa) a function *ST_VectorFromText(CHARACTER LARGE OBJECT)*,

ab) a function *ST_VectorFromText(CHARACTER LARGE OBJECT, INTEGER)*,

ac) a function *ST_VectorFromWKB(BINARY LARGE OBJECT)*,

ad) a function *ST_VectorFromWKB(BINARY LARGE OBJECT, INTEGER)*,

ae) a function *ST_VectorFromGML(CHARACTER LARGE OBJECT)*,

af) a function *ST_VectorFromGML(CHARACTER LARGE OBJECT, INTEGER)*,

ag) an ordering function *ST_OrderingEquals*(*ST_Vector, ST_Vector)*,

ah) an SQL Transform group *ST_WellKnownText*,

ai) an SQL Transform group *ST_WellKnownBinary*,

aj) an SQL Transform group *ST_GML*.

2) The *ST_PrivateX* attribute contains the x coordinate value.

3) The *ST_PrivateY* attribute contains the y coordinate value.

4) The *ST_PrivateZ* attribute contains the z coordinate value.

5) An *ST_Vector* value returned by the constructor function corresponds to the empty set.

6) An *ST_Vector* value is not well formed if either:

   a) the *ST_PrivateX* attribute is the null value and the *ST_PrivateY* attribute is not the null value,

   b) the *ST_PrivateY* attribute is the null value and the *ST_PrivateX* attribute is not the null value,

   c) *ST_Is3D* returns 0 (zero) and the *ST_PrivateZ* attribute is not the null value,

   d) *ST_Is3D* returns 1 (one), the *ST_PrivateX* attribute is the null value, and the *ST_PrivateZ* attribute is not the null value,

   e) *ST_Is3D* returns 1 (one), the *ST_PrivateX* attribute is not the null value, and the *ST_PrivateZ* attribute is the null value.

7) An *ST_Vector* value has an associated spatial reference system specified by a spatial reference system identifier.

8) The coordinate dimension shall be the same as the coordinate dimension of the spatial reference system for the *ST_Vector* value.

9) An *ST_Vector* value shall not have m coordinates.

17.2.2 ST_Vector Methods

Purpose

Return an ST_Vector value constructed from either the well-known text representation, the well-known binary representation, the GML representation, or the specified coordinate values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Vector
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxVectorAsText))
   RETURNS ST_Vector
   FOR ST_Vector
   RETURN NEW ST_Vector(awktorgml, 0)

CREATE CONSTRUCTOR METHOD ST_Vector
   (awktorgml CHARACTER LARGE OBJECT(ST_MaxVectorAsText),
    ansrid INTEGER)
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END

CREATE CONSTRUCTOR METHOD ST_Vector
   (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary))
   RETURNS ST_Vector
   FOR ST_Vector
   RETURN NEW ST_Vector(awkb, 0)

CREATE CONSTRUCTOR METHOD ST_Vector
   (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary),
    ansrid INTEGER)
   RETURNS ST_Vector
   FOR ST_Vector
   RETURN ST_VectorFromWKB(awkb, ansrid)

CREATE CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION)
   RETURNS ST_Vector
   FOR ST_Vector
   RETURN NEW ST_Vector(xcoord, ycoord, 0)

CREATE CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Vector
   FOR ST_Vector
   RETURN SELF.                    -- Return an ST_Vector value with
      ST_SRID(ansrid).                   -- SRID = ansrid,
      ST_X(xcoord).                     -- x coordinate = xcoord,
      ST_Y(ycoord)                      -- y coordinate = ycoord

CREATE CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION)
   RETURNS ST_Vector
   FOR ST_Vector
   RETURN NEW ST_Vector(xcoord, ycoord, zcoord, 0)
```

```
CREATE CONSTRUCTOR METHOD ST_Vector
   (xcoord DOUBLE PRECISION,
    ycoord DOUBLE PRECISION,
    zcoord DOUBLE PRECISION,
    ansrid INTEGER)
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      IF ( xcoord IS NULL AND ycoord IS NOT NULL ) OR
         ( xcoord IS NOT NULL AND ycoord IS NULL ) THEN
         -- if not well-formed, raise an exception
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      IF xcoord IS NULL THEN
         -- If xcoord is the null value, assume an empty
         -- point value is being created, check zcoord is null.
         IF zcoord IS NOT NULL THEN
            SIGNAL SQLSTATE '2FF16'
               SET MESSAGE_TEXT = 'not an empty set';
         END IF;
      ELSE
         -- Otherwise, check zcoord is not null.
         IF zcoord IS NULL THEN
            SIGNAL SQLSTATE '2FF03'
               SET MESSAGE_TEXT = 'null argument';
         END IF;
      END IF;
      RETURN SELF.                    -- Return an ST_Vector value with
         ST_SRID(ansrid).               -- SRID = ansrid,
         ST_PrivateX(xcoord).          -- x coordinate = xcoord,
         ST_PrivateY(ycoord).          -- y coordinate = ycoord,
         ST_PrivateZ(zcoord);          -- z coordinate = zcoord
   END
```

Definitional Rules

1) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.

2) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Vector* value.

Description

1) The method *ST_Vector(CHARACTER LARGE OBJECT)* takes the following input parameter:

   a) a CHARACTER LARGE OBJECT value *awktorgml*.

2) The null-call type-preserving SQL-invoked constructor method *ST_Vector(CHARACTER LARGE OBJECT)* returns the result of the value expression: *NEW ST_Vector(awktorgml, 0)*.

3) The method *ST_Vector(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a CHARACTER LARGE OBJECT value *awktorgml*,

   b) an INTEGER value *ansrid*.

4) For the null-call type-preserving SQL-invoked constructor method *ST_Vector(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) If *awktorgml* contains a Vector XML element in the GML representation, then return the result of the value expression: *ST_VectorFromGML(awktorgml, ansrid)*.

b) Otherwise, return the result of the value expression: *ST_VectorFromText(awktorgml, ansrid)*.

5) The method *ST_Vector(BINARY LARGE OBJECT)* takes the following input parameter:

   a) a BINARY LARGE OBJECT value *awkb*.

6) The null-call type-preserving SQL-invoked constructor method *ST_Vector(BINARY LARGE OBJECT)* returns the result of the value expression: *NEW ST_Vector(awkb, 0)*.

7) The method *ST_Vector(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:

   a) a BINARY LARGE OBJECT value *awkb*,

   b) an INTEGER value *ansrid*.

8) The null-call type-preserving SQL-invoked constructor method *ST_Vector(BINARY LARGE OBJECT, INTEGER)* returns the result of the value expression: *ST_VectorFromWKB(awkb, ansrid)*.

9) The method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *xcoord*,

   b) a DOUBLE PRECISION value *ycoord*.

10) The null-call type-preserving SQL-invoked constructor method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Vector(xcoord, ycoord, 0)*.

11) The method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) a DOUBLE PRECISION value *xcoord*,

   b) a DOUBLE PRECISION value *ycoord*,

   c) an INTEGER value *ansrid*.

12) The null-call type-preserving SQL-invoked constructor method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* returns an *ST_Vector* value with:

   a) The spatial reference system identifier set to *ansrid*.

   b) Using the method *ST_X(DOUBLE PRECISION)*, the x coordinate value is set to *xcoord*.

   c) Using the method *ST_Y(DOUBLE PRECISION)*, the y coordinate value is set to *ycoord*.

   d) The z coordinate value set to NULL by default clause.

13) The method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *xcoord*,

   b) a DOUBLE PRECISION value *ycoord*,

   c) a DOUBLE PRECISION value *zcoord*.

14) The null-call type-preserving SQL-invoked constructor method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION)* returns the result of the value expression: *NEW ST_Vector(xcoord, ycoord, zcoord, 0)*.

15) The method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)* takes the following input parameters:

   a) a DOUBLE PRECISION value *xcoord*,

   b) a DOUBLE PRECISION value *ycoord*,

   c) a DOUBLE PRECISION value *zcoord*,

   d) an INTEGER value *ansrid*.

16) For the type-preserving SQL-invoked constructor method *ST_Vector(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER)*:

Case:

a) If *xcoord* is the null value and ycoord is not the null value, or if *xcoord* is not the null value and ycoord is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

b) If *xcoord* is the null value and *zcoord* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set*.

c) If *xcoord* is not the null value and *zcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

d) Otherwise, return an *ST_Vector* value with:

   i) The spatial reference system identifier set to *ansrid*.

   ii) The x coordinate value is set to *xcoord*.

   iii) The y coordinate value is set to *ycoord*.

   iv) The z coordinate value is set to *zcoord*.

17.2.3 ST_X Methods

Purpose

Observe and mutate the x coordinate value of an ST_Vector value.

Definition

```
CREATE METHOD ST_X()
   RETURNS DOUBLE PRECISION
   FOR ST_Vector
   RETURN SELF.ST_PrivateX

CREATE METHOD ST_X
   (xcoord DOUBLE PRECISION)
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      IF xcoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         IF SELF IS NULL THEN
            RETURN CAST (NULL AS ST_Vector);
         END IF;
         RETURN
            SELF.ST_PrivateX(xcoord);
      END IF;
   END
```

Description

1) The method *ST_X()* has no input parameters.
2) The null-call method *ST_X()* returns the value of the *ST_PrivateX* attribute.
3) The method *ST_X(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *xcoord*.
4) For the type-preserving method *ST_X(DOUBLE PRECISION)*:

   Case:

   a) If *xcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateX(xcoord)*.

17.2.4 ST_Y Methods

Purpose

Observe and mutate the y coordinate value of an ST_Vector value.

Definition

```
CREATE METHOD ST_Y()
   RETURNS DOUBLE PRECISION
   FOR ST_Vector
   RETURN SELF.ST_PrivateY

CREATE METHOD ST_Y
   (ycoord DOUBLE PRECISION)
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      IF ycoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      ELSE
         IF SELF IS NULL THEN
            RETURN CAST (NULL AS ST_Vector);
         END IF;
         RETURN
            SELF.ST_PrivateY(ycoord);
      END IF;
   END
```

Description

1) The method *ST_Y()* has no input parameters.

2) The null-call method *ST_Y()* returns the value of the *ST_PrivateY* attribute.

3) The method *ST_Y(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *ycoord*.

4) For the type-preserving method *ST_Y(DOUBLE PRECISION)*:

   Case:

   a) If *ycoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateY(ycoord)*.

17.2.5 ST_Z Methods

Purpose

Observe and mutate the z coordinate value of an ST_Vector value.

Definition

```
CREATE METHOD ST_Z()
   RETURNS DOUBLE PRECISION
   FOR ST_Vector
   RETURN SELF.ST_PrivateZ

CREATE METHOD ST_Z
   (zcoord DOUBLE PRECISION)
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      IF SELF.ST_IsEmpty() = 0 AND zcoord IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      IF SELF.ST_IsEmpty() = 1 AND zcoord IS NOT NULL THEN
         SIGNAL SQLSTATE '2FF16'
            SET MESSAGE_TEXT = 'not an empty set';
      END IF;
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Vector);
      END IF;
      RETURN
         SELF.ST_PrivateZ(zcoord);
   END
```

Description

1) The method *ST_Z()* has no input parameters.

2) The null-call method *ST_Z()* returns the value of the *ST_PrivateZ* attribute.

3) The method *ST_Z(DOUBLE PRECISION)* takes the following input parameters:

   a) a DOUBLE PRECISION value *zcoord*.

4) For the type-preserving method *ST_Z(DOUBLE PRECISION)*:

   Case:

   a) If SELF is an empty set and *zcoord* is not the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument.*

   b) If SELF is not an empty set and *zcoord* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – not an empty set.*

   c) If SELF is the null value, then return the null value.

   d) Otherwise, return the result of the value expression: *SELF.ST_PrivateZ(zcoord)*.

17.2.6 ST_Coordinates Method

Purpose

Return the coordinate values as a DOUBLE PRECISION ARRAY value.

Definition

```
CREATE METHOD ST_Coordinates()
   RETURNS DOUBLE PRECISION ARRAY[3]
   FOR ST_Vector
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         WHEN (SELF.ST_Z() IS NOT NULL THEN
            ARRAY[SELF.ST_X(), SELF.ST_Y(), SELF.ST_Z()]
         ELSE
            ARRAY[SELF.ST_X(), SELF.ST_Y()]
      END
```

Description

1) The method *ST_Coordinates()* has no input parameters.

2) For the null-call method *ST_Coordinates()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) If the z coordinate value is not the null value, then return an array of type DOUBLE PRECISION with the first element representing the x coordinate value, the second element representing the y coordinate value, and the third element representing the z coordinate value.

   c) Otherwise, return an array of type DOUBLE PRECISION with the first element representing the x coordinate value and the second element representing the y coordinate value.

17.2.7 ST_Is3D Method

Purpose

Test if an ST_Vector value has z coordinate values.

Definition

```
CREATE METHOD ST_Is3D()
   RETURNS INTEGER
   FOR ST_Vector
   RETURN (SELF.ST_Z NOT NULL)
```

Description

1) The method *ST_Is3D()* has no input parameters.
2) The null-call method *ST_Is3D()* returns the value of (*SELF.ST_Z NOT NULL*).

17.2.8 ST_SRID Methods

Purpose

Observe and mutate the spatial reference system identifier of the ST_Vector value.

Definition

```
CREATE METHOD ST_SRID()
   RETURNS INTEGER
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END

CREATE METHOD ST_SRID
   (ansrid INTEGER)
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_SRID()* has no input parameters.
2) The null-call method *ST_SRID()* returns the spatial reference system identifier for the *ST_Vector* value.
3) The method *ST_SRID(INTEGER)* takes the following input parameters:
   a) an INTEGER value *ansrid*.
4) The parameter *ansrid* is a spatial reference system identifier.
5) The null-call type-preserving method *ST_SRID(INTEGER)* returns an *ST_Vector* value with the spatial reference system identifier set to *ansrid*.

17.2.9 ST_IsEmpty Method

Test if an ST_Vector value corresponds to the empty set.

Definition

```
CREATE METHOD ST_IsEmpty()
   RETURNS INTEGER
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsEmpty()* has no input parameters.

2) For the null-call method *ST_IsEmpty()*:

   Case:

   a) If the *ST_Vector* value corresponds to the empty set, then return 1 (one).

   b) Otherwise, return 0 (zero).

3) An *ST_Vector* value returned by the constructor function corresponds to the empty set.

17.2.10 ST_Equals Method

Purpose

Test if an ST_Vector value is equal to another ST_Vector value.

Definition

```
CREATE METHOD ST_Equals
   (avector ST_Vector)
   RETURNS INTEGER
   FOR ST_Vector
   RETURN ((SELF.ST_SRID() = avector.ST_SRID()) AND
     (SELF.ST_X() = avector.ST_X()) AND
     (SELF.ST_Y() = avector.ST_Y()) AND
     ((SELF.ST_Z() = avector.ST_Z()) OR
       (SELF.ST_Z() IS NULL AND avector.ST_Z() IS NULL)))
```

Description

1) The method *ST_Equals(ST_Vector)* takes the following input parameters:
   a) an *ST_Vector* value *avector*.
2) The null-call method *ST_Equals(ST_Vector)* returns the result of the value expression: ((*SELF.ST_SRID() = avector.ST_SRID()*) AND (*SELF.ST_X() = avector.ST_X()*) AND (*SELF.ST_Y() = avector.ST_Y()*) AND ((*SELF.ST_Z() = avector.ST_Z()) OR* (SELF.ST_Z() IS NULL AND avector.ST_Z() IS NULL))).

17.2.11 ST_WKTToSQL Method

Purpose

Return an ST_Vector value for a given well-known text representation.

Definition

```
CREATE METHOD ST_WKTToSQL
   (awkt CHARACTER LARGE OBJECT(ST_MaxVectorAsText))
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Vector* value.

Description

1) The method *ST_WKTToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The parameter *awkt* is the well-known text representation of an *ST_Vector* value. If *awkt* is not producible in the BNF for <well-known text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
3) The null-call method *ST_WKTToSQL(CHARACTER LARGE OBJECT)* returns an *ST_Vector* value represented by *awkt.*

17.2.12 ST_AsText Method

Purpose

Return the well-known text representation of an ST_Vector value.

Definition

```
CREATE METHOD ST_AsText()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxVectorAsText)
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Vector* value.

Description

1) The method *ST_AsText()* has no input parameters.
2) The null-call method *ST_AsText()* returns a CHARACTER LARGE OBJECT value containing the well-known text representation of SELF. Values shall be produced in the BNF for <well-known text representation>.

17.2.13 ST_WKBToSQL Method

Purpose

Return an ST_Vector value for a given well-known binary representation.

Definition

```
CREATE METHOD ST_WKBToSQL
   (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary))
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.

Description

1) The method *ST_WKBToSQL(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The parameter *awkb* is the well-known binary representation of an *ST_Vector* value. If *awkb* is not producible in the BNF for <well-known binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
3) The null-call method *ST_WKBToSQL(BINARY LARGE OBJECT)* returns an *ST_Vector* value represented by *awkb*.

17.2.14 ST_AsBinary Method

Purpose

Return the well-known binary representation of an ST_Vector value.

Definition

```
CREATE METHOD ST_AsBinary()
   RETURNS BINARY LARGE OBJECT(ST_MaxVectorAsBinary)
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.

Description

1) The method *ST_AsBinary()* has no input parameters.

2) The null-call method *ST_AsBinary()* returns a BINARY LARGE OBJECT value containing the well-known binary representation of SELF. Values shall be produced in the BNF for <well-known binary representation>.

17.2.15 ST_GMLToSQL Method

Purpose

Return an ST_Vector value for a given GML representation.

Definition

```
CREATE METHOD ST_GMLToSQL
   (agml CHARACTER LARGE OBJECT(ST_MaxVectorAsGML))
   RETURNS ST_Vector
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Vector* value.

Description

1) The method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The parameter *agml* is the GML representation of an *ST_Vector* value. If *agml* does not contain a Vector XML element , then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
3) The x coordinate value of an *ST_Vector* value is represented as the first coordinate value of a vector XML element.
4) The y coordinate value of an *ST_Vector* value is represented as the second coordinate value of a vector XML element.
5) The z coordinate value of an *ST_Vector* value is represented as the third coordinate value of a vector XML element.

   Case:

   a) If the vector XML element contains three coordinate values, then the resulting ST_Vector value will have x, y, and z coordinate values.
   b) Otherwise, the resulting *ST_Vector* value will have only x and y coordinate values.
6) Let *S* be the spatial reference system identifier for the resulting *ST_Vector* value.

   Case:

   a) If the srsname XML attribute is not specified, then set *S* to 0 (zero).
   b) Otherwise,

      Case:

      i) If the value of srsname XML attribute is producible in the BNF for <spatial reference system>, then
         1) Set *SRT* to the spatial reference system text in the srsname XML attribute.
         2) Select the row in the SPATIAL_REF_SYS view where the SRTEXT column is equal to *SRT*.

            Case:

            A) If the row is not found, then the following exception condition is raised: *SQL/MM Spatial Exception – unknown spatial reference system*.

B) Otherwise, set *S* to the value of the SRID column in the returned row.

ii) If the value of the srsname XML attribute is in the form: *ON*:*OI* where *ON* is the organization name and *OI* is organization assigned identifier, then

1) Let *AN* be the organization name *ON* and *AI* be the organization assigned identifier *OI*.

2) Select the row in the SPATIAL_REF_SYS view where the AUTH_NAME column is equal to *AN* and AUTH_ID column is equal to *AI*.

Case:

A) If the row is not found, then the following exception condition is raised: *SQL/MM Spatial Exception – unknown spatial reference system*.

B) Otherwise, set *S* to the value of the SRID column in the returned row.

iii) Otherwise, the following exception condition is raised: *SQL/MM Spatial Exception – unknown spatial reference system*.

7) The null-call method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* returns an *ST_Vector* value represented by *agml*.

17.2.16 ST_AsGML Method

Purpose

Return the GML representation of an ST_Vector value.

Definition

```
CREATE METHOD ST_AsGML()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxVectorAsGML)
   FOR ST_Vector
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Vector* value.

Description

1) The method *ST_AsGML()* has no input parameters.
2) The null-call method *ST_AsGML()* returns a CHARACTER LARGE OBJECT value containing a GML representation. The instantiable type ST_Vector is mapped to an XML Vector element in the GML representation.
3) The srsname XML attribute of the XML element identifies its spatial reference system. Select the row in the SPATIAL_REF_SYS view where the *srid* is equal to *SELF.ST_SRID()*. For the selected row, let *AN* be the value of the AUTH_NAME column, *AI* be the value of the AUTH_ID column and *SRT* be the value of the SRTEXT column.

   Case:

   a) If the *AN* is not the null value and *AI* is not the null value then the srsname XML attribute attribute is specified as:

      srsname='*AN*:*AI*'

   b) Otherwise, the srsname XML attribute is specified as:

      srsname='*SRT*'

17.2.17 ST_VectorFromText Functions

Purpose

Return an ST_Vector value which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Vector value.

Definition

```
CREATE FUNCTION ST_VectorFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxVectorAsText))
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_VectorFromText(awkt, 0)

CREATE FUNCTION ST_VectorFromText
   (awkt CHARACTER LARGE OBJECT(ST_MaxVectorAsText),
    ansrid INTEGER)
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Vector* value.

Description

1) The function *ST_VectorFromText(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*.
2) The null-call function *ST_VectorFromText(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_VectorFromText(awkt, 0)*.
3) The function *ST_VectorFromText(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *awkt*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_VectorFromText(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkt* is the well-known text representation of an *ST_Vector* value. If *awkt* is not producible in the BNF for <vector text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.
   b) Otherwise, return an ST_Vector value represented by *awkt* with the spatial reference system identifier set to *ansrid*.

17.2.18 ST_VectorFromWKB Functions

Purpose

Return an ST_Vector value which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Vector value.

Definition

```
CREATE FUNCTION ST_VectorFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary))
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_VectorFromWKB(awkb, 0)
```

```
CREATE FUNCTION ST_VectorFromWKB
   (awkb BINARY LARGE OBJECT(ST_MaxVectorAsBinary),
    ansrid INTEGER)
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.

Description

1) The function *ST_VectorFromWKB(BINARY LARGE OBJECT)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*.
2) The null-call function *ST_VectorFromWKB(BINARY LARGE OBJECT)* returns the result of the value expression: *ST_VectorFromWKB(awkb, 0)*.
3) The function *ST_VectorFromWKB(BINARY LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a BINARY LARGE OBJECT value *awkb*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_VectorFromWKB(BINARY LARGE OBJECT, INTEGER)*:

   Case:

   a) The parameter *awkb* is the well-known binary representation of an *ST_Vector* value. If *awkb* is not producible in the BNF for <vector binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.
   b) Otherwise, return an ST_Vector value represented by *awkb* with the spatial reference system identifier set to *ansrid*.

17.2.19 ST_VectorFromGML Functions

Purpose

Return an ST_Vector value which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Vector.

Definition

```
CREATE FUNCTION ST_VectorFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxVectorAsGML))
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURN ST_VectorFromGML(agml, 0)

CREATE FUNCTION ST_VectorFromGML
   (agml CHARACTER LARGE OBJECT(ST_MaxVectorAsGML),
    ansrid INTEGER)
   RETURNS ST_Vector
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxVectorAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Vector* value.

Description

1) The function *ST_VectorFromGML(CHARACTER LARGE OBJECT)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*.
2) The null-call function *ST_VectorFromGML(CHARACTER LARGE OBJECT)* returns the result of the value expression: *ST_VectorFromGML(agml, 0)*.
3) The function *ST_VectorFromGML(CHARACTER LARGE OBJECT, INTEGER)* takes the following input parameters:
   a) a CHARACTER LARGE OBJECT value *agml*,
   b) an INTEGER value *ansrid*.
4) For the null-call function *ST_VectorFromGML(CHARACTER LARGE OBJECT, INTEGER)*:

   Case:
   a) If the parameter *agml* does not contain a Vector XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.
   b) Return an *ST_Vector* value represented by *agml* with the spatial reference system identifier set to *ansrid*.
   c) The x coordinate value of an *ST_Vector* value is represented as the first coordinate value of a Vector XML element.

d) The y coordinate value of an *ST_Vector* value is represented as the second coordinate value of a Vector XML element.

e) The z coordinate value of an *ST_Vector* value is represented as the third coordinate value of a Vector XML element.

   Case:

   i) If the vector XML element contains three coordinate values, then the resulting *ST_Vector* value will have x, y, and z coordinate values.

   ii) Otherwise, the resulting *ST_Vector* value will have only x and y coordinate values.

f) If the Vector XML element does not specify a spatial reference system, *asrid* is set to NULL.

17.2.20 ST_Vector Ordering Definition

Purpose

Provide the equals only ordering definition for the ST_Vector type.

Definition

```
CREATE FUNCTION ST_OrderingEquals
   (avector ST_Vector,
    anothervector ST_Vector)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN
      CASE
         WHEN avector.ST_Equals(anothervector) = 1 THEN
            0
         ELSE
            1
      END

CREATE ORDERING FOR ST_Vector
   EQUALS ONLY BY RELATIVE WITH
      FUNCTION ST_OrderingEquals(ST_Vector, ST_Vector)
```

Description

1) The function *ST_OrderingEquals(ST_Vector, ST_Vector)* takes the following input parameters:
   a) an *ST_Vector* value *avector*,
   b) an *ST_Vector* value *anothervector*.
2) For the null-call function *ST_OrderingEquals(ST_Vector, ST_Vector)*:

   Case:
   a) If the value expression: *avector.ST_Equals(anothervector)* is 1 (one), then return 0 (zero).
   b) Otherwise, return 1 (one)
3) Use the function *ST_OrderingEquals(ST_Vector, ST_Vector)* to define ordering for the *ST_Vector* type.

17.2.21 SQL Transform Functions

Purpose

Define SQL transform functions for the ST_Vector type.

Definition

```
CREATE TRANSFORM FOR ST_Vector
   ST_WellKnownText
      (TO SQL WITH METHOD ST_WKTToSQL
         (CHARACTER LARGE OBJECT(ST_MaxVectorAsText)),
       FROM SQL WITH METHOD ST_AsText())
   ST_WellKnownBinary
      (TO SQL WITH METHOD ST_WKBToSQL
         (BINARY LARGE OBJECT(ST_MaxVectorAsBinary)),
       FROM SQL WITH METHOD ST_AsBinary())
   ST_GML
      (TO SQL WITH METHOD ST_GMLToSQL
         (CHARACTER LARGE OBJECT(ST_MaxVectorAsGML)),
       FROM SQL WITH METHOD ST_AsGML())
```

Definitional Rules

1) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_Vector* value.
2) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.
3) *ST_MaxVectorAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Vector* value.

Description

1) Use the method *ST_WKTToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsText()* to define the transform group *ST_WellKnownText*.
2) Use the method *ST_WKBToSQL(BINARY LARGE OBJECT)* and the method ST_AsBinary*()* to define the transform group *ST_WellKnownBinary*.
3) Use the method *ST_GMLToSQL(CHARACTER LARGE OBJECT)* and the method *ST_AsGML()* to define the transform group *ST_GML*.

17.2.22 <well-known text representation>

Purpose

This subclause contains the definition of the <well-known text representation> of an ST_Vector value.

Description

1) The well-known text representation of an *ST_Vector* value is defined by the following BNF for <vector text representation>.

```
<vector text representation> ::=
     VECTOR [ <just z> ] <vector text>

<vector text> ::=
     <empty set>
   | <left paren> <vector> <right paren>

<vector> ::= <x> <y> [ <z> ]

<x> ::= <number>

<y> ::= <number>

<z> ::= <number>

<empty set> ::= EMPTY

<just z> ::=
     Z

<left paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<right paren> ::=
     !! See Subclause 5.1, "<SQL terminal character>", in
        Part 2 of ISO/IEC 9075

<number> ::=
     <exact numeric literal>
   | <approximate numeric literal>

<exact numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075

<approximate numeric literal> ::=
     !! See Subclause 5.3, "<literal>", in Part 2 of ISO/IEC 9075
```

a) <vector text representation> is the well-known text representation for an *ST_Vector* value that is produced by <vector text>.

b) Case:

   i) If <just z> is specified, then let *ZORM* be Z.

   ii) Otherwise, let *ZORM* be 2D.

c) Case:

   i) If <vector text> immediately contains an <empty set>, then:

      Case:

      1) If *ZORM* is Z, then <vector text> produces an empty set of type *ST_Vector* as the result of the value expression: *NEW ST_Vector(NULL, NULL, NULL)*.

      2) Otherwise, <vector text> produces an empty set of type *ST_Vector* as the result of the value expression: *NEW ST_Vector()*.

   ii) Otherwise, <vector text> produces the *ST_Vector* value from <vector>.

d) Let *XC* be the DOUBLE PRECISION value specified by <x> in <vector> and *YC* be the DOUBLE PRECISION value specified by <y> in <vector>.

   Case:

   i) If *ZORM* is Z then,

      1) If <vector> does not contain <z>, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

      2) Let *ZC* be the DOUBLE PRECISION value specified by <z> in <vector>.

      3) <vector> produces an *ST_Vector* value as the result of the value expression: *NEW ST_Vector(XC, YC, ZC)*.

   ii) Otherwise,

      1) If <vector> contains <z>, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

      2) <vector> produces an *ST_Vector* value as the result of the value expression: *NEW ST_Vector(XC, YC)*.

e) The list of keywords is EMPTY, VECTOR, and Z.

17.2.23 <well-known binary representation>

Purpose

This subclause contains the definition of <well-known binary representation>.

Description

1) The well-known binary representation of an *ST_Vector* value is defined by the following BNF for <well-known binary representation>.

```
<well-known binary representation> ::=
     <vectorz binary representation>
   | <vector binary representation>

<vectorz binary representation> ::=
     <byte order> <wkbvectorz> [ <wkbvectorz binary> ]

<vector binary representation> ::=
     <byte order> <wkbvector> [ <wkbvector binary> ]

<wkbvectorz binary> ::= <wkbx> <wkby> <wkbz>

<wkbvector binary> ::= <wkbx> <wkby>

<wkbx> ::= <double>

<wkby> ::= <double>

<wkbz> ::= <double>

<wkbvectorz> ::= <uint32>

<wkbvector> ::= <uint32>

<byte order> ::=
     <big endian>
   | <little endian>

<big endian> ::= !! See Description

<little endian> ::= !! See Description

<uint32> ::= !! See Description

<double> ::= !! See Description
```

a) Case:

i) If <well-known binary representation> immediately contains a <vectorz binary representation>, then <well-known binary representation> produces an ST_Vector value specified by the immediately contained <vectorz binary representation>.

ii) Otherwise, <well-known binary representation> produces an ST_Vector value specified by the immediately contained <vector binary representation>.

b) Case:

i) If <vectorz binary representation> immediately contains a <wkbvectorz binary>, then <vectorz binary representation> is the well-known binary representation for an *ST_Vector* value that is produced by <wkbvectorz binary>.

ii) Otherwise, <vectorz binary representation> produces an empty set of type *ST_Vector.*

c) Case:

i) If <vector binary representation> immediately contains a <wkbvector binary>, then <vector binary representation> is the well-known binary representation for an *ST_Vector* value that is produced by <wkbvector binary>.

ii) Otherwise, <vector binary representation> produces an empty set of type *ST_Vector.*

d) Let *XC* be the DOUBLE PRECISION value specified by <wkbx> in <wkbvectorz binary>, *YC* be the DOUBLE PRECISION value specified by <wkby> in <wkbvectorz binary>, and *ZC* be the DOUBLE PRECISION value specified by <wkbz> in <wkbvectorz binary>. <wkbvectorz binary> produces an *ST_Vector* value as the result of the value expression: *NEW ST_Vector(XC, YC, ZC*).

e) Let *XC* be the DOUBLE PRECISION value specified by <wkbx> in <wkbvector binary> and *YC* be the DOUBLE PRECISION value specified by <wkby> in <wkbvector binary>. <wkbvector binary> produces an *ST_Vector* value as the result of the value expression: *NEW ST_Vector(XC, YC)*.

f) The <well-known binary representation> <uint32> values for <wkbvector> and <wkbvectorz> are 101 and 1101, respectively.

g) <byte order> indicates the binary representation of <uint32> and <double> values that follow <byte order>.

h) <big endian> is a <byte order> represented by a <byte> with the value 0 (zero). <uint32> is Big Endian (most significant octet first). <double> is Big Endian (sign bit is in the first octet).

i) <little endian> is a <byte order> represented by a <byte> with the value 1 (one). <uint32> is Little Endian (most significant octet last). <double> is Little Endian (sign bit is in the last octet).

j) <uint32> s a 32 bit (4 octets) data type that encodes an unsigned integer in the range [0, 4294967295].

k) <double> is a 64 bit (8 octets) double precision data type that encodes a double precision format using the IEC 559:1989.

l) <well-known binary representation> provides a portable representation of a geometry value as a contiguous stream of octets in a BINARY LARGE OBJECT value. The serialized *ST_Vector* is either represented in Big Endian format or Little Endian format. Conversion between Big Endian format or Little Endian format is a simple operation involving reversing the order of octets within each <uint32> or <double> value in the BINARY LARGE OBJECT.

17.3 ST_AffinePlacement Type and Routines

17.3.1 ST_AffinePlacement Type

Purpose

The ST_AffinePlacement type defines a linear transformation from a constructive parameter space to the coordinate space of the coordinate reference system being used.

Definition

```
CREATE TYPE ST_AffinePlacement
   AS (
      ST_PrivateLocation ST_Point DEFAULT NULL,
      ST_PrivateReferenceDirections
        ST_Vector ARRAY[ST_MaxVectorArrayElements] DEFAULT ARRAY []
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_AffinePlacement
      (location ST_Point,
       referencedirectionarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
      RETURNS ST_AffinePlacement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Location()
      RETURNS ST_Point
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Location
      (location ST_Point)
      RETURNS ST_AffinePlacement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_RefDirections()
      RETURNS ST_Vector ARRAY[ST_MaxVectorArrayElements]
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_RefDirections
      (referencedirectionarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
      RETURNS ST_AffinePlacement
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
```

```
METHOD ST_InDimension()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_OutDimension()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_Transform
   (apoint ST_Point)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,

METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.
2) The attribute *ST_PrivateLocation* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateLocation*.
3) The attribute *ST_PrivateReferenceDirections* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateReferenceDirections*.

Description

1) The *ST_AffinePlacement* type provides for public use:
   a) a method *ST_AffinePlacement(ST_Point, ST_Vector ARRAY)*,
   b) a method *ST_Location()*,
   c) a method *ST_Location(ST_Point)*,
   d) a method *ST_RefDirections()*,
   e) a method *ST_RefDirections(ST_Vector ARRAY)*,
   f) a method *ST_InDimension()*,
   g) a method *ST_OutDimension()*,
   h) a method *ST_Transform(ST_Point)*,
   i) a method *ST_IsEmpty()*.
2) The *ST_PrivateLocation* attribute contains the location value which is the target of the parameter space origin.
3) The *ST_PrivateReferenceDirections* attribute contains the reference directions value which is the target directions for the coordinate basis vectors of the parameter space.

4) An *ST_AffinePlacement* value returned by the constructor function corresponds to the empty set.
5) The coordinate dimension of the *ST_Point* value of the *ST_PrivateLocation* attribute shall be equal to the coordinate dimension of all of the *ST_Vector* values in the *ST_PrivateReferenceDirections* attribute.
6) *ST_IsMeasured* for the *ST_Point* value of the *ST_PrivateLocation* attribute shall be equal to 0 (zero).
7) The spatial reference system of the *ST_Point* value of the *ST_PrivateLocation* attribute shall be the same spatial reference system as all of the *ST_Vector* values in the *ST_PrivateReferenceDirections* attribute.
8) The cardinality of the *ST_Vector* ARRAY *ST_PrivateReferenceDirections* attribute value shall be equal to the coordinate dimension of the parameter space being transformed.

17.3.2 ST_AffinePlacement Method

Purpose

Return an ST_AffinePlacement value constructed from the ST_Point location and ST_Vector ARRAY reference directions values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_AffinePlacement
   (location ST_Point,
    referencedirectionarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   RETURNS ST_AffinePlacement
   FOR ST_AffinePlacement
   RETURN SELF.                 -- Return an ST_AffinePlacement value with
      ST_Location(location).     -- location = location,
      ST_RefDirections(referencedirectionarray) -- refdirections =
               referencedirectionarray
```

Definitional Rules

1) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

Description

1) The method *ST_AffinePlacement(ST_Point, ST_Vector ARRAY)* takes the following input parameter:
   a) an *ST_Point* value *location*.
   b) an *ST_Vector* ARRAY value *referencedirectionarray*.
2) The null-call type-preserving SQL-invoked constructor method *ST_AffinePlacement(ST_Point, ST_Vector ARRAY)* returns an *ST_AffinePlacement* value with:
   a) Using the method *ST_Location(ST_Point)*, the location value is set to *location*.
   b) Using the method *ST_RefDirections(ST_Vector ARRAY)*, the refdirections value is set to *referencedirectionarray*.

17.3.3 ST_Location Methods

Purpose

Observe and mutate the ST_Point location attribute of the ST_AffinePlacement value which is the target of the parameter space origin.

Definition

```
CREATE METHOD ST_Location()
   RETURNS ST_Point
   FOR ST_AffinePlacement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateLocation
      END

CREATE METHOD ST_Location
   (location ST_Point)
   RETURNS ST_AffinePlacement
   FOR ST_AffinePlacement
   BEGIN
      -- If location is the null value, then raise an exception
      IF location IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_AffinePlacement);
      END IF;
      -- location (control) point should not be measured.
      IF location.ST_IsMeasured() = 1 THEN
         SIGNAL SQLSTATE '2FF97'
            SET MESSAGE_TEXT = 'm coordinates not allowed';
      END IF;
      -- If reference directions exist, then location must agree with them
      -- in SRID and coordinate dimension
      IF CARDINALITY(SELF.ST_RefDirections()) > 0 THEN
         IF location.ST_SRID() <> ST_CheckSRID(SELF.ST_RefDirections())
            THEN SIGNAL SQLSTATE '2FF10'
               SET MESSAGE_TEXT = 'mixed spatial reference systems';
         END IF;
         IF location.ST_CoordDim() <>
           ST_GetCoordDim(SELF.ST_RefDirections())
            THEN SIGNAL SQLSTATE '2FF25'
               SET MESSAGE_TEXT = 'mixed coordinate dimensions';
         END IF;
      END IF;
      RETURN
         SELF.ST_PrivateLocation(location);
   END
```

Description

1) The method *ST_Location()* has no input parameters.

2) For the null-call method *ST_Location()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateLocation* attribute of SELF.

3) The method *ST_Location(ST_Point)* takes the following input parameters:

   a) an *ST_Point* value *location*.

4) For the type-preserving method *ST_Location(ST_Point)*:

   Case:

   a) If *location* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) If *location* is measured, then an exception condition is raised: *SQL/MM Spatial exception – m coordinates not allowed*.

   d) If *SELF.ST_PrivateReferenceDirections* already has *ST_Vector elements*, then:

      i) Using the procedure *ST_CheckSRID(SELF.ST_RefDirections()),* verify that all *ST_Vectors* have the same SRID. If this SRID value is not equal to the SRID of *location,* then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

      ii) Using the procedure *ST_GetCoordDim(SELF.ST_RefDirections()),* verify that all *ST_Vectors* have the same coordinate dimension. If this coordinate dimension value is not equal to the coordinate dimension of *location,* then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

   e) Otherwise, return the result of the value expression: *SELF.ST_PrivateLocation(location)*.

17.3.4 ST_RefDirections Methods

Purpose

Observe and mutate the collection of ST_Vectors that is the reference directions attribute of an ST_AffinePlacement value.

Definition

```
CREATE METHOD ST_RefDirections()
   RETURNS ST_Vector ARRAY[ST_MaxVectorArrayElements]
   FOR ST_AffinePlacement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateReferenceDirections
      END

CREATE METHOD ST_RefDirections
   (referencedirectionarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   RETURNS ST_AffinePlacement
   FOR ST_AffinePlacement
   BEGIN
      -- If referencedirectionarray is the null value, raise an exception
      IF referencedirectionarray IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
      SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_AffinePlacement);
      END IF;
      -- reference directions vectors must all agree with location
      -- in SRID and coordinate dimension
      IF SELF.ST_Location().ST_SRID() <>
        ST_CheckSRID(referencedirectionarray)
         THEN SIGNAL SQLSTATE '2FF10'
            SET MESSAGE_TEXT = 'mixed spatial reference systems';
      END IF;
      IF SELF.ST_Location().ST_CoordDim() <>
        ST_GetCoordDim(referencedirectionarray)
         THEN SIGNAL SQLSTATE '2FF25'
            SET MESSAGE_TEXT = 'mixed coordinate dimensions';
      END IF;
      RETURN
         SELF.ST_PrivateReferenceDirections(referencedirectionarray)
   END
```

Definitional Rules

1) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

Description

1) The method *ST_RefDirections()* has no input parameters.
2) For the null-call method *ST_RefDirections()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateReferenceDirections* attribute of SELF.

3) The method *ST_RefDirections(ST_Vector ARRAY)* takes the following input parameters:

   a) an *ST_Vector* ARRAY value *referencedirectionarray*.

4) For the type-preserving method *ST_RefDirections(ST_Vector ARRAY)*:

   Case:

   a) If *referencedirectionarray* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If SELF is the null value, then return the null value.

   c) Using the procedure *ST_CheckSRID(referencedirectionarray),* verify that all *ST_Vectors* have the same SRID. If this SRID value is not equal to the SRID of *SELF.ST_Location(),* then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

   d) Using the procedure *ST_GetCoordDim(referencedirectionarray),* verify that all *ST_Vectors* have the same coordinate dimension. If this coordinate dimension value is not equal to the coordinate dimension of *SELF.ST_Location(),* then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

   e) Otherwise, return the result of the value expression: *SELF.ST_PrivateReferenceDirections(referencedirectionarray)*.

17.3.5 ST_InDimension Method

Purpose

Return the INTEGER in dimension value of an ST_AffinePlacement value as the dimension of the input parameter space which is equal to the number of reference directions.

Definition

```
CREATE METHOD ST_InDimension()
   RETURNS INTEGER
   FOR ST_AffinePlacement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            CARDINALITY(SELF.ST_RefDirections)
      END
```

Description

1) The method *ST_InDimension()* has no input parameters.

2) For the null-call method *ST_InDimension()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the cardinality of the *ST_Vector* ARRAY that is the *ST_RefDirections* attribute of SELF.

17.3.6 ST_OutDimension Method

Purpose

Return the INTEGER out dimension value of an ST_AffinePlacement value as the dimension of the output coordinate reference system which is equal to the dimension of the reference directions.

Definition

```
CREATE METHOD ST_OutDimension()
   RETURNS INTEGER
   FOR ST_AffinePlacement
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            ST_GetCoordDim(SELF.ST_PrivateReferenceDirections)
      END
```

Description

1) The method *ST_OutDimension()* has no input parameters.

2) For the null-call method *ST_OutDimension()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the result of the value expression: *ST_GetCoordDim(SELF.ST_PrivateReferenceDirections)*.

17.3.7 ST_Transform Method

Purpose

Map the parameter coordinate point to the corresponding coordinate point in the output Cartesian space.

Definition

```
CREATE METHOD ST_Transform
   (apoint ST_Point)
   RETURNS ST_Point
   FOR ST_AffinePlacement
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_Transform(ST_Point)* takes the following input parameters:
   a) an *ST_Point* value *apoint*.
2) The null-call method *ST_Transform(ST_Point)* returns the *ST_Point* value that is the coordinate point in the output Cartesian space corresponding to the parameter coordinate point *apoint*.

17.3.8 ST_IsEmpty Method

Test if an ST_AffinePlacement value corresponds to the empty set.

Definition

```
CREATE METHOD ST_IsEmpty()
   RETURNS INTEGER
   FOR ST_AffinePlacement
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsEmpty()* has no input parameters.

2) For the null-call method *ST_IsEmpty()*:

   Case:

   a) If the *ST_AffinePlacement* value corresponds to the empty set, then return 1 (one).

   b) Otherwise, return 0 (zero).

3) An *ST_AffinePlacement* value returned by the constructor function corresponds to the empty set.

17.4 ST_NURBSPoint Type and Routines

17.4.1 ST_NURBSPoint Type

Purpose

The ST_NURBSPoint type defines a is a NURBS control point which has been adjusted to consider its respective weight value.

Definition

```
CREATE TYPE ST_NURBSPoint
   AS (
      ST_PrivateWeightedPoint ST_Point DEFAULT NULL,
      ST_PrivateWeight DOUBLE PRECISION DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_NURBSPoint
      (weightedpoint ST_Point,
       weight DOUBLE PRECISION)
      RETURNS ST_NURBSPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_WeightedPoint()
      RETURNS ST_Point
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_WeightedPoint
      (weightedpoint ST_Point)
      RETURNS ST_NURBSPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_Weight()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Weight
      (weight DOUBLE PRECISION)
      RETURNS ST_NURBSPoint
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
```



```
METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateWeightedPoint* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateWeightedPoint*.
2) The attribute *ST_PrivateWeight* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateWeight* .

Description

1) The *ST_NURBSPoint* type provides for public use:
   a) a method *ST_NURBSPoint(ST_Point, DOUBLE PRECISION)*,
   b) a method *ST_WeightedPoint()*,
   c) a method *ST_WeightedPoint(ST_Point)*,
   d) a method *ST_Weight()*,
   e) a method *ST_Weight(DOUBLE PRECISION)*,
   f) a method *ST_IsEmpty()*.
2) The *ST_PrivateWeightedPoint*attribute contains the weighted ST_Point value whose coordinate values include consideration of the weight value.
3) The *ST_PrivateWeight* attribute contains the divisor for the rational spline control point. For rational curves, all control points must have weight values.
4) An *ST_NURBSPoint* value returned by the constructor function corresponds to the empty set.

17.4.2 ST_NURBSPoint Method

Purpose

Return an ST_NURBSPoint value constructed from the ST_Point weighted point and DOUBLE PRECISION weight values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_NURBSPoint
   (weightedpoint ST_Point,
    weight DOUBLE PRECISION)
   RETURNS ST_NURBSPoint
   FOR ST_NURBSPoint
   RETURN SELF.              -- Return an ST_NURBSPoint value with
      ST_WeightedPoint(weightedpoint).  -- weightedpoint = weightedpoint,
      ST_Weight(weight)            -- weight = weight
```

Description

1) The method *ST_NURBSPoint(ST_Point, DOUBLE PRECISION)* takes the following input parameter:
   a) an *ST_Point* value *weightedpoint*.
   b) a DOUBLE PRECISION value *weight*.
2) The null-call type-preserving SQL-invoked constructor method *ST_NURBSPoint(ST_Point, DOUBLE PRECISION)* returns an *ST_NURBSPoint* value with:
   a) Using the method *ST_WeightedPoint(ST_Point)*, the weightedpoint value is set to *weightedpoint*.
   b) Using the method *ST_Weight(DOUBLE PRECISION)*, the weight value is set to *weight*.

17.4.3 ST_WeightedPoint Methods

Purpose

Observe and mutate the ST_Point weighted point attribute of the ST_NURBSPoint value which contains the weighted ST_Point value whose coordinate values include consideration of the weight value.

Definition

```
CREATE METHOD ST_WeightedPoint()
   RETURNS ST_Point
   FOR ST_NURBSPoint
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateWeightedPoint
      END

CREATE METHOD ST_WeightedPoint
   (weightedpoint ST_Point)
   RETURNS ST_NURBSPoint
   FOR ST_NURBSPoint
   BEGIN
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_NURBSPoint);
      END IF;
      RETURN
         SELF.ST_PrivateWeightedPoint(weightedpoint);
   END
```

Description

1) The method *ST_WeightedPoint()* has no input parameters.

2) For the null-call method *ST_WeightedPoint()*:

   Case:

   a) If SELF is an empty set, then return the null value.

   b) Otherwise, return the *ST_PrivateWeightedPoint* attribute of SELF.

3) The method *ST_WeightedPoint(ST_Point)* takes the following input parameters:

   a) an *ST_Point* value *weightedpoint*.

4) For the type-preserving method *ST_WeightedPoint(ST_Point)*:

   Case:

   a) If SELF is the null value, then return the null value.

   b) Otherwise, return the result of the value expression: *SELF.ST_PrivateWeightedPoint(weightedpoint)*.

17.4.4 ST_Weight Methods

Purpose

Observe and mutate the DOUBLE PRECISION weight attribute of the ST_NURBSPoint value which contains the divisor for the rational spline control point.

Definition

```
CREATE METHOD ST_Weight()
   RETURNS DOUBLE PRECISION
   FOR ST_NURBSPoint
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateWeight
      END

CREATE METHOD ST_Weight
   (weight DOUBLE PRECISION)
   RETURNS ST_NURBSPoint
   FOR ST_NURBSPoint
   BEGIN
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_NURBSPoint);
      END IF;
      RETURN
         SELF.ST_PrivateWeight(weight)
   END
```

Description

1) The method *ST_Weight()* has no input parameters.
2) For the null-call method *ST_Weight()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateWeight* attribute of SELF.
3) The method *ST_Weight(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *weight*.
4) For the type-preserving method *ST_Weight(DOUBLE PRECISION)*:

   Case:

   a) If SELF is the null value, then return the null value.
   b) Otherwise, return the result of the value expression: *SELF.ST_PrivateWeight(weight)*.

17.4.5 ST_IsEmpty Method

Test if an ST_NURBSPoint value corresponds to the empty set.

Definition

```
CREATE METHOD ST_IsEmpty()
   RETURNS INTEGER
   FOR ST_NURBSPoint
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsEmpty()* has no input parameters.
2) For the null-call method *ST_IsEmpty()*:

   Case:

   a) If the *ST_NURBSPoint* value corresponds to the empty set, then return 1 (one).
   b) Otherwise, return 0 (zero).
3) An *ST_NURBSPoint* value returned by the constructor function corresponds to the empty set.

17.5 ST_Knot Type and Routines

17.5.1 ST_Knot Type

Purpose

The ST_Knot type represents a knot value and the number of times that value occurs (multiplicity) in the ST_NURBSCurve knot sequence

Definition

```
CREATE TYPE ST_Knot
   AS (
      ST_PrivateValue DOUBLE PRECISION DEFAULT NULL,
      ST_PrivateMultiplicity INTEGER DEFAULT NULL
   )
   INSTANTIABLE
   NOT FINAL

   CONSTRUCTOR METHOD ST_Knot
      (value DOUBLE PRECISION,
       multiplicity INTEGER)
      RETURNS ST_Knot
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Value()
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Value
      (value DOUBLE PRECISION)
      RETURNS ST_Knot
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD ST_Multiplicity()
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD ST_Multiplicity
      (multiplicity INTEGER)
      RETURNS ST_Knot
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
```

```
METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

Definitional Rules

1) The attribute *ST_PrivateValue* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateValue*.
2) The attribute *ST_PrivateMultiplicity* is not for public use. There are no GRANT statements granting EXECUTE privilege on the observer or mutator method for *ST_PrivateMultiplicity*.

Description

1) The *ST_Knot* type provides for public use:
   a) a method *ST_Knot(DOUBLE PRECISION, INTEGER)*,
   b) a method *ST_Value()*,
   c) a method *ST_Value(DOUBLE PRECISION)*,
   d) a method *ST_Multiplicity()*,
   e) a method *ST_Multiplicity(INTEGER)*,
   f) a method *ST_IsEmpty()*.
2) The *ST_PrivateValue* attribute contains the DOUBLE PRECISION value of the knot.
3) The *ST_PrivateMultiplicity* attribute contains the INTEGER number of times that the knot value occurs for the specified curve.
4) An *ST_Knot* value returned by the constructor function corresponds to the empty set.

17.5.2 ST_Knot Method

Purpose

Return an ST_Knot value constructed from the DOUBLE PRECISION value and INTEGER multiplicity values.

Definition

```
CREATE CONSTRUCTOR METHOD ST_Knot
   (value DOUBLE PRECISION,
    multiplicity INTEGER)
   RETURNS ST_Knot
   FOR ST_Knot
   RETURN SELF.              -- Return an ST_Knot value with
      ST_Value(value).              -- value = value,
      ST_Multiplicity(multiplicity)      -- multiplicity = multiplicity
```

Description

1) The method *ST_Knot(DOUBLE PRECISION, INTEGER)* takes the following input parameter:
   a) a DOUBLE PRECISION value *value*.
   b) an INTEGER value *multiplicity*.
2) The null-call type-preserving SQL-invoked constructor method *ST_Knot(DOUBLE PRECISION, INTEGER)* returns an *ST_Knot* value with:
   a) Using the method *ST_Value(DOUBLE PRECISION)*, the value value is set to *value*.
   b) Using the method *ST_Multiplicity(INTEGER)*, the multiplicity value is set to *multiplicity*.

17.5.3 ST_Value Methods

Purpose

Observe and mutate the DOUBLE PRECISION value attribute of the ST_Knot value which contains the value of the knot.

Definition

```
CREATE METHOD ST_Value()
   RETURNS DOUBLE PRECISION
   FOR ST_Knot
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateValue
      END

CREATE METHOD ST_Value
   (value DOUBLE PRECISION)
   RETURNS ST_Knot
   FOR ST_Knot
   BEGIN
      -- If value is the null value, then raise an exception
      IF value IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Knot);
      END IF;
      RETURN
         SELF.ST_PrivateValue(value);
   END
```

Description

1) The method *ST_Value()* has no input parameters.
2) For the null-call method *ST_Value()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateValue* attribute of SELF.
3) The method *ST_Value(DOUBLE PRECISION)* takes the following input parameters:
   a) a DOUBLE PRECISION value *value*.
4) For the type-preserving method *ST_Value(DOUBLE PRECISION)*:

   Case:

   a) If *value* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateValue(value)*.

17.5.4 ST_Multiplicity Methods

Purpose

Observe and mutate the INTEGER multiplicity attribute of the ST_Knot value which contains the number of times that the knot value occurs for the specified curve.

Definition

```
CREATE METHOD ST_Multiplicity()
   RETURNS INTEGER
   FOR ST_Knot
   RETURN
      CASE
         WHEN SELF.ST_IsEmpty() = 1 THEN
            NULL
         ELSE
            SELF.ST_PrivateMultiplicity
      END

CREATE METHOD ST_Multiplicity
   (multiplicity INTEGER)
   RETURNS ST_Knot
   FOR ST_Knot
   BEGIN
      -- If multiplicity is the null value, then raise an exception
      IF multiplicity IS NULL THEN
         SIGNAL SQLSTATE ‘2FF03’
            SET MESSAGE_TEXT = ‘null argument’;
      END IF;
      -- If SELF is the null value, then return the null value.
      IF SELF IS NULL THEN
         RETURN CAST (NULL AS ST_Knot);
      END IF;
      RETURN
         SELF.ST_PrivateMultiplicity(multiplicity)
   END
```

Description

1) The method *ST_Multiplicity()* has no input parameters.
2) For the null-call method *ST_Multiplicity()*:

   Case:

   a) If SELF is an empty set, then return the null value.
   b) Otherwise, return the *ST_PrivateMultiplicity* attribute of SELF.
3) The method *ST_Multiplicity(INTEGER)* takes the following input parameters:
   a) an INTEGER value *multiplicity*.
4) For the type-preserving method *ST_Multiplicity(INTEGER)*:

   Case:

   a) If *multiplicity* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELF is the null value, then return the null value.
   c) Otherwise, return the result of the value expression: *SELF.ST_PrivateMultiplicity(multiplicity)*.

17.5.5 ST_IsEmpty Method

Test if an ST_Knot value corresponds to the empty set.

Definition

```
CREATE METHOD ST_IsEmpty()
   RETURNS INTEGER
   FOR ST_Knot
   BEGIN
      --
      -- See Description
      --
   END
```

Description

1) The method *ST_IsEmpty()* has no input parameters.
2) For the null-call method *ST_IsEmpty()*:

   Case:

   a) If the *ST_Knot* value corresponds to the empty set, then return 1 (one).
   b) Otherwise, return 0 (zero).
3) An *ST_Knot* value returned by the constructor function corresponds to the empty set.

18 Support Routines

18.1 ST_Geometry ARRAY Support Routines

18.1.1 ST_MaxDimension Function

Purpose

Return the maximum geometric dimension value in an ST_Geometry ARRAY value.

Definition

```
CREATE FUNCTION ST_MaxDimension
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE maxdimension SMALLINT;
      DECLARE counter INTEGER;

      -- If the array is empty, then -1
      -- (the dimension of an empty set)
      IF CARDINALITY(ageometryarray) = 0 THEN
         RETURN -1;
      -- Otherwise,
      ELSE
         SET counter = 1;
         -- For each element in ageometryarray
         WHILE counter <= CARDINALITY(ageometryarray) DO
            -- If the current element is the first element, then
            -- set maxdimension to the dimension of the current value.
            IF counter = 1 THEN
               SET maxdimension = ageometryarray[counter].ST_Dimension();
            -- Otherwise, if the dimension of the current value is
            -- greater than maxdimension, set maxdimension to the
            -- dimension of the current value.
            ELSEIF ageometryarray[counter].ST_Dimension() >
               maxdimension THEN
               SET maxdimension = ageometryarray[counter].ST_Dimension();
            END IF;
            SET counter = counter + 1;
         END WHILE;
         -- Return the maximum dimension
         RETURN maxdimension;
      END IF;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_MaxDimension(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the null-call function *ST_MaxDimension(ST_Geometry ARRAY)*:

   Case:

a) If the cardinality of *ageometryarray* is equal to 0 (zero), return -1.

b) Otherwise,

   i) For the elements in *ageometryarray*:

      Case:

      1) If the current element is the first element, then let *maxdimension* be the dimension of the current element.

      2) Otherwise, if the dimension of the current element is greater than *maxdimension*, then let *maxdimension* be the dimension of the current element.

   ii) Return *maxdimension*.

18.1.2 ST_CheckSRID Function

Purpose

If the elements in the ST_Geometry ARRAY or ST_Vector ARRAY value have mixed spatial reference systems, then raise an exception. Otherwise, return the spatial reference system identifier of the ST_Geometry or ST_Vector elements.

Definition

```
CREATE FUNCTION ST_CheckSRID
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE srid INTEGER;

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      SET srid = 0;
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         IF counter = 1 THEN
            SET srid = ageometryarray[counter].ST_SRID();
         ELSEIF srid <> ageometryarray[counter].ST_SRID() THEN
            SIGNAL SQLSTATE '2FF10'
               SET MESSAGE_TEXT = 'mixed spatial reference systems';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      RETURN srid;
   END

CREATE FUNCTION ST_CheckSRID
   (avectorarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE srid INTEGER;

      -- If avectorarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(avectorarray);
      SET srid = 0;
      SET counter = 1;
      -- For each element in avectorarray
      WHILE counter <= CARDINALITY(avectorarray) DO
         IF counter = 1 THEN
            SET srid = avectorarray[counter].ST_SRID();
         ELSEIF srid <> avectorarray[counter].ST_SRID() THEN
```

```
            SIGNAL SQLSTATE '2FF10'
               SET MESSAGE_TEXT = 'mixed spatial reference systems';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      RETURN srid;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

Description

1) The function *ST_CheckSRID(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the function *ST_CheckSRID(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If the cardinality of *ageometryarray* is 0 (zero), then return 0 (zero).

      ii) If any two elements of *ageometryarray* are not in the same spatial reference system, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

      iii) Otherwise, return the spatial reference system identifier common to all elements in *ageometryarray*.

3) The function *ST_CheckSRID(ST_Vector ARRAY)* takes the following input parameters:

   a) an *ST_Vector* ARRAY value *avectorarray*.

4) For the function *ST_CheckSRID(ST_VectorARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Vector ARRAY)* to check if *avectorarray* is the null value or contains null elements.

   b) Case:

      i) If the cardinality of *avectorarray* is 0 (zero), then return 0 (zero).

      ii) If any two elements of *avectorarray* are not in the same spatial reference system, then an exception condition is raised: *SQL/MM Spatial exception – mixed spatial reference systems*.

      iii) Otherwise, return the spatial reference system identifier common to all elements in *avectorarray*.

18.1.3 ST_GetCoordDim Functions

Purpose

Return the coordinate dimension value in an ST_Geometry ARRAY value, ST_Vector ARRAY value or ST_NURBSPoint ARRAY value.

Definition

```
CREATE FUNCTION ST_GetCoordDim
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE coorddim SMALLINT;
      DECLARE is3d SMALLINT;
      DECLARE ismeasured SMALLINT;
      DECLARE counter INTEGER;

      -- If the array is empty, then 2 (the default)
      IF CARDINALITY(ageometryarray) = 0 THEN
         RETURN 2;
      -- Otherwise,
      ELSE
         SET counter = 1;
         -- For each element in ageometryarray
         WHILE counter <= CARDINALITY(ageometryarray) DO
            -- If the current element is the first element, then
            -- set coorddim to the coordinate dimension of the
            -- current value.
            IF counter = 1 THEN
               BEGIN
                  SET coorddim = ageometryarray[counter].ST_CoordDim();
                  SET is3d = ageometryarray[counter].ST_Is3D();
                  SET ismeasured =
                  ageometryarray[counter].ST_IsMeasured();
               END;
            -- Otherwise, if the coordinate dimension of the current
            -- value is not equal to coorddim, raise an exception.
            ELSEIF ageometryarray[counter].ST_Is3D() <> is3d OR
               ageometryarray[counter].ST_IsMeasured() <> ismeasured THEN
               SIGNAL SQLSTATE '2FF25'
                  SET MESSAGE_TEXT = 'mixed coordinate dimensions';
            END IF;
            SET counter = counter + 1;
         END WHILE;
         -- Return the common coordinate dimension
         RETURN coorddim;
      END IF;
   END
```

```
CREATE FUNCTION ST_GetCoordDim
   (ageometry ST_Geometry,
    ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN ST_GetCoordDim(ARRAY [ ageometry ] || ageometryarray)

CREATE FUNCTION ST_GetCoordDim
   (avectorarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE coorddim SMALLINT;
      DECLARE is3d SMALLINT;
      DECLARE counter INTEGER;

      -- If the array is empty, then 2 (the default)
      IF CARDINALITY(avectorarray) = 0 THEN
         RETURN 2;
      -- Otherwise,
      ELSE
         SET counter = 1;
         -- For each element in avectorarray
         WHILE counter <= CARDINALITY(avectorarray) DO
            -- If the current element is the first element, then
            -- set coorddim to the coordinate dimension of the
            -- current value.
            IF counter = 1 THEN
               BEGIN
                  SET coorddim = 2
                  IF avectorarray[counter].ST_Is3D()THEN
                     coorddim = coorddim + 1;
                  SET is3d = avectorarray[counter].ST_Is3D();
               END;
            -- Otherwise, if the coordinate dimension of the current
            -- value is not equal to coorddim, raise an exception.
            ELSEIF avectorarray[counter].ST_Is3D() <> is3d THEN
               SIGNAL SQLSTATE '2FF25'
                  SET MESSAGE_TEXT = 'mixed coordinate dimensions';
            END IF;
            SET counter = counter + 1;
         END WHILE;
         -- Return the common coordinate dimension
         RETURN coorddim;
      END IF;
   END

CREATE FUNCTION ST_GetCoordDim
   (anurbspointarray ST_NURBSPoint ARRAY[ST_MaxNURBSPointArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

```
STATIC DISPATCH
BEGIN
   DECLARE coorddim SMALLINT;
   DECLARE is3d SMALLINT;
   DECLARE counter INTEGER;

   -- If the array is empty, then 2 (the default)
   IF CARDINALITY(anurbspointarray) = 0 THEN
      RETURN 2;
   -- Otherwise,
   ELSE
      SET counter = 1;
      -- For each element in anurbspointarray
      WHILE counter <= CARDINALITY(anurbspointarray) DO
         -- If the current element is the first element, then
         -- set coorddim to the coordinate dimension of the
         -- current element’s weighted point ST_Point value.
         IF counter = 1 THEN
            BEGIN
               SET coorddim = 2
               IF
               anurbspointarray[counter].ST_WeightedPoint.ST_Is3D()THEN
                  coorddim = coorddim + 1;
               SET is3d =
                  anurbspointarray[counter].ST_WeightedPoint.ST_Is3D();
            END;
         -- Otherwise, if the coordinate dimension of the current
         -- value is not equal to coorddim, raise an exception.
         ELSEIF anurbspointarray[counter].ST_Is3D() <> is3d THEN
            SIGNAL SQLSTATE '2FF25'
               SET MESSAGE_TEXT = 'mixed coordinate dimensions';
         END IF;
         SET counter = counter + 1;
      END WHILE;
      -- Return the common coordinate dimension
      RETURN coorddim;
   END IF;
END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

2) *ST_MaxNURBSPointArrayElements* is the implementation-defined maximum cardinality of an array of *ST_NURBSPoint* values.

3) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

Description

1) The function *ST_GetCoordDim(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_GetCoordDim(ST_Geometry ARRAY)*:

   Case:

   a) If the cardinality of *ageometryarray* is equal to 0 (zero), return 2.

   b) Otherwise,

      i) For the elements in *ageometryarray*:

         Case:

1) If the current element is the first element, then let *COORDDIM* be the coordinate dimension of the current element, *IS3D* be the *ST_Is3D()* value of the current element and *ISMEASURED* be the *ST_IsMeasured()* value of the current element.

2) Otherwise, if *IS3D* is not equal to the *ST_Is3D()* value of the current element or *ISMEASURED* is not equal to the *ST_IsMeasured()* value of the current element, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

ii) Return *COORDDIM*.

3) The function *ST_GetCoordDim(ST_Geometry, ST_Geometry ARRAY)* takes the following input parameters:

a) an *ST_Geometry* value *ageometry*.

b) an *ST_Geometry* ARRAY value *ageometryarray*.

4) The null-call function *ST_GetCoordDim(ST_Geometry, ST_Geometry ARRAY)* returns the result of the value expression: *ST_GetCoordDim(ARRAY [ageometry] || ageometryarray).*

5) The function *ST_GetCoordDim(ST_Vector ARRAY)* takes the following input parameters:

a) an ST_Vector ARRAY value avectoryarray.

6) For the null-call function *ST_GetCoordDim(ST_Vector ARRAY)*:

Case:

a) If the cardinality of *avectoryarray* is equal to 0 (zero), return 2.

b) Otherwise,

i) For the elements in *avectoryarray*:

Case:

1) If the current element is the first element, then let *COORDDIM* be the coordinate dimension of the current element and let *IS3D* be the *ST_Is3D()* value of the current element.

2) Otherwise, if *IS3D* is not equal to the *ST_Is3D()* value of the current element, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

ii) Return *COORDDIM*.

7) The function *ST_GetCoordDim(ST_NURBSPoint ARRAY)* takes the following input parameters:

a) an ST_NURBSPoint ARRAY value anurbspointyarray.

8) For the null-call function ST_GetCoordDim(ST_NURBSPoint ARRAY):

Case:

a) If the cardinality of *anurbspointyarray* is equal to 0 (zero), return 2.

b) Otherwise,

i) For the elements in *anurbspointyarray*:

Case:

1) If the current element is the first element, then let *COORDDIM* be the coordinate dimension of the current element's weighted point ST_Point value and let *IS3D* be the *ST_Is3D()* value of the current element's weighted point ST_Point value.

2) Otherwise, if *IS3D* is not equal to the *ST_Is3D()* value of the current element's weighted point ST_Point value, then an exception condition is raised: *SQL/MM Spatial exception – mixed coordinate dimensions*.

ii) Return *COORDDIM*.

18.1.4 ST_GetIs3D Function

Purpose

Return the value for the ST_Is3D method which is consistent across all the ST_Geometry values in an ST_Geometry ARRAY value.

Definition

```
CREATE FUNCTION ST_GetIs3D
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN
      CASE
         WHEN ST_GetCoordDim(ageometryarray) = 2 THEN
            0
         ELSE
            ageometryarray[1].ST_Is3D()
      END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_GetIs3D(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the null-call function *ST_GetIs3D(ST_Geometry ARRAY)*:

   Case:
   a) If *ST_GetCoordDim(ageometryarray)* is equal to 2, return 0 (zero).
   b) Otherwise, return the result of the value expression: *ageometryarray[1].ST_Is3D().*

18.1.5 ST_GetIsMeasured Function

Purpose

Return the value for the ST_IsMeasured method which is consistent across all the ST_Geometry values in an ST_Geometry ARRAY value.

Definition

```
CREATE FUNCTION ST_GetIsMeasured
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS SMALLINT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   RETURN
      CASE
         WHEN ST_GetCoordDim(ageometryarray) = 2 THEN
            0
         ELSE
            ageometryarray[1].ST_IsMeasured()
      END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_GetIsMeasured(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the null-call function *ST_GetIsMeasured(ST_Geometry ARRAY)*:

   Case:
   a) If *ST_GetCoordDim(ageometryarray)* is equal to 2, return 0 (zero).
   b) Otherwise, return the result of the value expression: *ageometryarray[1].ST_IsMeasured().*

18.1.6 ST_CheckNulls Procedure

Purpose

Raise an exception if an ST_Geometry ARRAY or ST_Vector ARRAY value is the null value or contains null or empty elements.

Definition

```
CREATE PROCEDURE ST_CheckNulls
   (IN ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      DECLARE counter INTEGER;

      -- If ageometryarray is the null value, then raise an exception.
      IF ageometryarray IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is the null value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NULL THEN
            SIGNAL SQLSTATE '2FF09'
               SET MESSAGE_TEXT = 'element is a null value';
         END IF;
         IF ageometryarray[counter].ST_IsEmpty() = 1 THEN
            SIGNAL SQLSTATE '2FF06'
               SET MESSAGE_TEXT = 'element is an empty set';
         END IF;
         SET counter = counter + 1;
      END WHILE;
   END

CREATE PROCEDURE ST_CheckNulls
   (IN avectorarray ST_Vector ARRAY[ST_MaxVectorArrayElements])
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      DECLARE counter INTEGER;

      -- If avectorarray is the null value, then raise an exception.
      IF avectorarray IS NULL THEN
         SIGNAL SQLSTATE '2FF03'
            SET MESSAGE_TEXT = 'null argument';
      END IF;
      SET counter = 1;
      -- For each element in avectorarray
      WHILE counter <= CARDINALITY(avectorarray) DO
         -- If the current element is the null value, then
         -- raise an exception.
         IF avectorarray[counter] IS NULL THEN
            SIGNAL SQLSTATE '2FF09'
               SET MESSAGE_TEXT = 'element is a null value';
```

```
        END IF;
        IF avectorarray[counter].ST_IsEmpty() = 1 THEN
           SIGNAL SQLSTATE '2FF06'
              SET MESSAGE_TEXT = 'element is an empty set';
        END IF;
        SET counter = counter + 1;
     END WHILE;
  END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.
2) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

Description

1) The procedure *ST_CheckNulls(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the procedure *ST_CheckNulls(ST_Geometry ARRAY)*:

   Case:

   a) If *ageometryarray* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If any element of *ageometryarray* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – element is a null value*.
   c) If any element of *ageometryarray* is an empty set, then an exception condition is raised: *SQL/MM Spatial exception – element is an empty set*.
3) The procedure *ST_CheckNulls(ST_Vector ARRAY)* takes the following input parameters:
   a) an *ST_Vector* ARRAY value *avectorarray*.
4) For the procedure *ST_CheckNulls(ST_Vector ARRAY)*:

   Case:

   a) If *avectorarray* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null* argument.
   b) If any element of *avectorarray* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – element is a null value*.
   c) If any element of *avectorarray* is an empty set, then an exception condition is raised: *SQL/MM Spatial exception – element is an empty set*.

18.1.7 ST_CheckConsecDups Procedure

Purpose

Raise an exception if an ST_Geometry ARRAY value has consecutive duplicate values.

Definition

```
CREATE PROCEDURE ST_CheckConsecDups
   (IN ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      DECLARE counter INTEGER;

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      SET counter = 1;
      WHILE counter <= CARDINALITY(ageometryarray)-1 DO
         -- If the current element is equal to the next element, then
         -- raise an exception.
         IF ageometryarray[counter] = ageometryarray[counter+1] THEN
            SIGNAL SQLSTATE '2FF05'
               SET MESSAGE_TEXT = 'duplicate value';
         END IF;
         SET counter = counter + 1;
      END WHILE;
   END
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The procedure *ST_CheckConsecDups(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the procedure *ST_CheckConsecDups(ST_Geometry ARRAY)*:
   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.
   b) If any two consecutive *ST_Geometry* values in *ageometryarray* are equal, then an exception condition is raised: *SQL/MM Spatial exception – duplicate value*.

18.1.8 ST_ToPointAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Point valued elements to an ST_Point ARRAY value.

Definition

```
CREATE FUNCTION ST_ToPointAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Point ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE apointarray ST_Point ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set apointarray to an empty array.
      SET apointarray = CAST(ARRAY[] AS
         ST_Point ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Point value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Point) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT = 'element is not an ST_Point type';
         END IF;
         -- Cast the current element as an ST_Point and
         -- concatenate it to the end of apointarray.
         SET apointarray = apointarray ||
            CAST(ageometryarray[counter] AS ST_Point);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Point array
      RETURN apointarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Point ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToPointAry
      (ST_Point ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToPointAry(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the null-call function *ST_ToPointAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

i) If any element of *ageometryarray* is not an *ST_Point* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Point type*.

ii) Otherwise, return an *ST_Point* ARRAY value containing each element of *ageometryarray* cast as an *ST_Point* value.

3) Use the function *ST_ToPointAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Point* ARRAY value.

18.1.9 ST_ToCurveAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Curve valued elements to an ST_Curve ARRAY value.

Definition

```
CREATE FUNCTION ST_ToCurveAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Curve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE acurvearray ST_Curve ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set acurvearray to an empty array.
      SET acurvearray = CAST(ARRAY[] AS
         ST_Curve ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Curve value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Curve) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT = 'element is not an ST_Curve type';
         END IF;
         -- Cast the current element as an ST_Curve and
         -- concatenate it to the end of acurvearray.
         SET acurvearray = acurvearray ||
            CAST(ageometryarray[counter] AS ST_Curve);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Curve array
      RETURN acurvearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCurveAry
      (ST_Curve ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToCurveAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToCurveAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

i) If any element of *ageometryarray* is not an *ST_Curve* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Curve type*.

ii) Otherwise, return an *ST_Curve* ARRAY value containing each element of *ageometryarray* Cast as an *ST_Curve* value.

3) Use the function *ST_ToCurveAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Curve* ARRAY value.

18.1.10 ST_ToLineStringAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_LineString valued elements to an ST_LineString ARRAY value.

Definition

```
CREATE FUNCTION ST_ToLineStringAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE alinestringarray ST_LineString
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set alinestringarray to an empty array.
      SET alinestringarray = CAST(ARRAY[] AS
         ST_LineString ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_LineString value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_LineString) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT = 'element is not an ST_LineString type';
         END IF;
         -- Cast the current element as an ST_LineString and
         -- concatenate it to the end of alinestringarray.
         SET alinestringarray = alinestringarray ||
            CAST(ageometryarray[counter] AS ST_LineString);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_LineString array
      RETURN alinestringarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_LineString ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToLineStringAry
      (ST_LineString ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToLineStringAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToLineStringAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

   i) If any element of *ageometryarray* is not an *ST_LineString* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_LineString type*.

   ii) Otherwise, return an *ST_LineString* ARRAY value containing each element of *ageometryarray* cast as an *ST_LineString* value.

3) Use the function *ST_ToLineStringAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_LineString* ARRAY value.

18.1.11 ST_ToCircularAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_CircularString valued elements to an ST_CircularString ARRAY value.

Definition

```
CREATE FUNCTION ST_ToCircularAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CircularString ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE acircularstringarray ST_CircularString
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set acircularstringarray to an empty array.
      SET acircularstringarray = CAST(ARRAY[] AS
         ST_CircularString ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_CircularString value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_CircularString) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_CircularString type';
         END IF;
         -- Cast the current element as an ST_CircularString and
         -- concatenate it to the end of acircularstringarray.
         SET acircularstringarray = acircularstringarray ||
            CAST(ageometryarray[counter] AS ST_CircularString);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_CircularString array
      RETURN acircularstringarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_CircularString ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCircularAry
      (ST_CircularString ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToCircularAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToCircularAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_CircularString* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_CircularString type*.

      ii) Otherwise, return an *ST_CircularString* ARRAY value containing each element of *ageometryarray* cast as an *ST_CircularString* value.

3) Use the function *ST_ToCircularAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_CircularString* ARRAY value.

18.1.12 ST_ToCircleAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Circle valued elements to an ST_Circle ARRAY value.

Definition

```
CREATE FUNCTION ST_ToCircleAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Circle ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE acirclearray ST_Circle
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set acirclearray to an empty array.
      SET acirclearray = CAST(ARRAY[] AS
         ST_Circle ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Circle value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Circle) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_Circle type';
         END IF;
         -- Cast the current element as an ST_Circle and
         -- concatenate it to the end of acirclearray.
         SET acirclearray = acirclearray ||
            CAST(ageometryarray[counter] AS ST_Circle);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Circle array
      RETURN acirclearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Circle ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCircleAry
      (ST_Circle ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToCircleAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToCircleAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_Circle* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Circle type*.

      ii) Otherwise, return an *ST_Circle* ARRAY value containing each element of *ageometryarray* cast as an *ST_Circle* value.

3) Use the function *ST_ToCircleAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Circle* ARRAY value.

18.1.13 ST_ToGeodesicAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_GeodesicString valued elements to an ST_GeodesicString ARRAY value.

Definition

```
CREATE FUNCTION ST_ToGeodesicAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_GeodesicString ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE ageodesicarray ST_GeodesicString
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set ageodesicarray to an empty array.
      SET ageodesicarray = CAST(ARRAY[] AS
         ST_GeodesicString ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_GeodesicString value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_GeodesicString) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_GeodesicString type';
         END IF;
         -- Cast the current element as an ST_GeodesicString and
         -- concatenate it to the end of ageodesicarray.
         SET ageodesicarray = ageodesicarray ||
            CAST(ageometryarray[counter] AS ST_GeodesicString);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_GeodesicString array
      RETURN ageodesicarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_GeodesicString ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToGeodesicAry
      (ST_GeodesicString ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToGeodesicAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToGeodesicAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_GeodesicString* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_GeodesicString type*.

      ii) Otherwise, return an *ST_GeodesicString* ARRAY value containing each element of *ageometryarray* cast as an *ST_GeodesicString* value.

3) Use the function *ST_ToGeodesicAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_GeodesicString* ARRAY value.

18.1.14 ST_ToEllipticalAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_EllipticalCurve valued elements to an ST_EllipticalCurve ARRAY value.

Definition

```
CREATE FUNCTION ST_ToEllipticalAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_EllipticalCurve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE anellipticalarray ST_EllipticalCurve
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set anellipticalarray to an empty array.
      SET anellipticalarray = CAST(ARRAY[] AS
         ST_EllipticalCurve ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_EllipticalCurve value,
         then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_EllipticalCurve) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_EllipticalCurve type';
         END IF;
         -- Cast the current element as an ST_EllipticalCurve and
         -- concatenate it to the end of anellipticalarray.
         SET anellipticalarray = anellipticalarray ||
            CAST(ageometryarray[counter] AS ST_EllipticalCurve);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_EllipticalCurve array
      RETURN anellipticalarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_EllipticalCurve ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToEllipticalAry
      (ST_EllipticalCurve ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToEllipticalAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToEllipticalAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_EllipticalCurve* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_EllipticalCurve type*.

      ii) Otherwise, return an *ST_EllipticalCurve* ARRAY value containing each element of *ageometryarray* cast as an *ST_EllipticalCurve* value.

3) Use the function *ST_ToEllipticalAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_EllipticalCurve* ARRAY value.

18.1.15 ST_ToNURBSAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_NURBSCurve valued elements to an ST_NURBSCurve ARRAY value.

Definition

```
CREATE FUNCTION ST_ToNURBSAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_NURBSCurve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE anurbsarray ST_NURBSCurve
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set anurbsarray to an empty array.
      SET anurbsarray = CAST(ARRAY[] AS
         ST_NURBSCurve ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_NURBSCurve value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_NURBSCurve) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_NURBSCurve type';
         END IF;
         -- Cast the current element as an ST_NURBSCurve and
         -- concatenate it to the end of anurbsarray.
         SET anurbsarray = anurbsarray ||
            CAST(ageometryarray[counter] AS ST_NURBSCurve);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_NURBSCurve array
      RETURN anurbsarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_NURBSCurve ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToNURBSAry
      (ST_NURBSCurve ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToNURBSAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToNURBSAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_NURBSCurve* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_NURBSCurve type*.

      ii) Otherwise, return an *ST_NURBSCurve* ARRAY value containing each element of *ageometryarray* cast as an *ST_NURBSCurve* value.

3) Use the function *ST_ToNURBSAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_NURBSCurve* ARRAY value.

18.1.16 ST_ToClothoidAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Clothoid valued elements to an ST_Clothoid ARRAY value.

Definition

```
CREATE FUNCTION ST_ToClothoidAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Clothoid ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE aclothoidarray ST_Clothoid
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set aclothoidarray to an empty array.
      SET aclothoidarray = CAST(ARRAY[] AS
         ST_Clothoid ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Clothoid value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Clothoid) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_Clothoid type';
         END IF;
         -- Cast the current element as an ST_Clothoid and
         -- concatenate it to the end of aclothoidarray.
         SET aclothoidarray = aclothoidarray ||
            CAST(ageometryarray[counter] AS ST_Clothoid);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Clothoid array
      RETURN aclothoidarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Clothoid ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToClothoidAry
      (ST_Clothoid ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToClothoidAry(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToClothoidAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_Clothoid* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Clothoid type*.

      ii) Otherwise, return an *ST_Clothoid* ARRAY value containing each element of *ageometryarray* cast as an *ST_Clothoid* value.

3) Use the function *ST_ToClothoidAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Clothoid* ARRAY value.

18.1.17 ST_ToSpiralAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_SpiralCurve valued elements to an ST_SpiralCurve ARRAY value.

Definition

```
CREATE FUNCTION ST_ToSpiralAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_SpiralCurve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE aspiralarray ST_SpiralCurve
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set aspiralarray to an empty array.
      SET aspiralarray = CAST(ARRAY[] AS
         ST_SpiralCurve ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_SpiralCurve value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_SpiralCurve) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_SpiralCurve type';
         END IF;
         -- Cast the current element as an ST_SpiralCurve and
         -- concatenate it to the end of aspiralarray.
         SET aspiralarray = aspiralarray ||
            CAST(ageometryarray[counter] AS ST_SpiralCurve);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_SpiralCurve array
      RETURN aspiralarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_SpiralCurve ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToSpiralAry
      (ST_SpiralCurve ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToSpiralAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToSpiralAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_SpiralCurve* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_SpiralCurve type*.

      ii) Otherwise, return an *ST_SpiralCurve* ARRAY value containing each element of *ageometryarray* cast as an *ST_SpiralCurve* value.

3) Use the function *ST_ToSpiralAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_SpiralCurve* ARRAY value.

18.1.18 ST_ToCompoundAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_CompoundCurve valued elements to an ST_CompoundCurve ARRAY value.

Definition

```
CREATE FUNCTION ST_ToCompoundAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundCurve ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE acompoundcurvearray ST_CompoundCurve
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set acompoundcurvearray to an empty array.
      SET acompoundcurvearray = CAST(ARRAY[] AS
         ST_CompoundCurve ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_CompoundCurve value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_CompoundCurve) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_CompoundCurve type';
         END IF;
         -- Cast the current element as an ST_CompoundCurve and
         -- concatenate it to the end of acompoundcurvearray.
         SET acompoundcurvearray = acompoundcurvearray ||
            CAST(ageometryarray[counter] AS ST_CompoundCurve);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_CompoundCurve array
      RETURN acompoundcurvearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_CompoundCurve ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCompoundAry
      (ST_CompoundCurve ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToCompoundAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToCompoundAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_CompoundCurve* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_CompoundCurve type*.

      ii) Otherwise, return an *ST_CompoundCurve* ARRAY value containing each element of *ageometryarray* cast as an *ST_CompoundCurve* value.

3) Use the function *ST_ToCompoundAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_CompoundCurve* ARRAY value.

18.1.19 ST_ToSurfaceAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Surface valued elements to an ST_Surface ARRAY value.

Definition

```
CREATE FUNCTION ST_ToSurfaceAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE asurfacearray ST_Surface ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set asurfacearray to an empty array.
      SET asurfacearray = CAST(ARRAY[] AS
         ST_Surface ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Surface value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Surface) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT = 'element is not an ST_Surface type';
         END IF;
         -- Cast the current element as an ST_Surface and
         -- concatenate it to the end of asurfacearray.
         SET asurfacearray = asurfacearray ||
            CAST(ageometryarray[counter] AS ST_Surface);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Surface array
      RETURN asurfacearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToSurfaceAry
      (ST_Surface ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToSurfaceAry(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the null-call function *ST_ToSurfaceAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

i) If any element of *ageometryarray* is not an *ST_Surface* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Surface type*.

ii) Otherwise, return an *ST_Surface* ARRAY value containing each element of *ageometryarray* cast as an *ST_Surface* value.

3) Use the function *ST_ToSurfaceAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Surface* ARRAY value.

18.1.20 ST_ToCurvePolyAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_CurvePolygon valued elements to an ST_CurvePolygon ARRAY value.

Definition

```
CREATE FUNCTION ST_ToCurvePolyAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CurvePolygon ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE acurvepolygonarray ST_CurvePolygon
      ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set acurvepolygonarray to an empty array.
      SET acurvepolygonarray = CAST(ARRAY[] AS
         ST_CurvePolygon ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_CurvePolygon value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_CurvePolygon) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT =
                  'element is not an ST_CurvePolygon type';
         END IF;
         -- Cast the current element as an ST_CurvePolygon and
         -- concatenate it to the end of acurvepolygonarray.
         SET acurvepolygonarray = acurvepolygonarray ||
            CAST(ageometryarray[counter] AS ST_CurvePolygon);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_CurvePolygon array
      RETURN acurvepolygonarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_CurvePolygon ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCurvePolyAry
      (ST_CurvePolygon ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToCurvePolyAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToCurvePolyAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_CurvePolygon* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_CurvePolygon type*.

      ii) Otherwise, return an *ST_CurvePolygon* ARRAY value containing each element of *ageometryarray* cast as an *ST_CurvePolygon* value.

3) Use the function *ST_ToCurvePolyAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_CurvePolygon* ARRAY value.

18.1.21 ST_ToPolygonAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Polygon valued elements to an ST_Polygon ARRAY value.

Definition

```
CREATE FUNCTION ST_ToPolygonAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Polygon ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE apolygonarray ST_Polygon ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set apolygonarray to an empty array.
      SET apolygonarray = CAST(ARRAY[] AS
         ST_Polygon ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Polygon value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Polygon) THEN
            SIGNAL SQLSTATE '2FF08'
               SET MESSAGE_TEXT = 'element is not an ST_Polygon type';
         END IF;
         -- Cast the current element as an ST_Polygon and
         -- concatenate it to the end of apolygonarray.
         SET apolygonarray = apolygonarray ||
            CAST(ageometryarray[counter] AS ST_Polygon);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Polygon array
      RETURN apolygonarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToPolygonAry
      (ST_Polygon ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToPolygonAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToPolygonAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

i) If any element of *ageometryarray* is not an *ST_Polygon* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Polygon type*.

ii) Otherwise, return an *ST_Polygon* ARRAY value containing each element of *ageometryarray* cast as an *ST_Polygon* value.

3) Use the function *ST_ToPolygonAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Polygon* ARRAY value.

18.1.22 ST_ToTriangleAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_Triangle valued elements to an ST_Triangle ARRAY value.

Definition

```
CREATE FUNCTION ST_ToTriangleAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Triangle ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE atrianglearray ST_Triangle
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set atrianglearray to an empty array.
      SET atrianglearray = CAST(ARRAY[] AS
         ST_Triangle ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_Triangle value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_Triangle) THEN
            SIGNAL SQLSTATE '2FF68'
               SET MESSAGE_TEXT = 'element is not an ST_Triangle type';
         END IF;
         -- Cast the current element as an ST_Triangle and
         -- concatenate it to the end of atrianglearray.
         SET atrianglearray = atrianglearray ||
            CAST(ageometryarray[counter] AS ST_Triangle);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_Triangle array
      RETURN atrianglearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_Triangle ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToTriangleAry
      (ST_Triangle ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToTriangleAry(ST_Geometry ARRAY)* takes the following input parameters:
   a) an *ST_Geometry* ARRAY value *ageometryarray*.
2) For the null-call function *ST_ToTriangleAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

  i) If any element of *ageometryarray* is not an *ST_Triangle* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_Triangle type*.

  ii) Otherwise, return an *ST_Triangle* ARRAY value containing each element of *ageometryarray* cast as an *ST_Triangle* value.

3) Use the function *ST_ToTriangleAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_Triangle* ARRAY value.

18.1.23 ST_ToPolyhdrlAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_PolyhdrlSurface valued elements to an ST_PolyhdrlSurface ARRAY value.

Definition

```
CREATE FUNCTION ST_ToPolyhdrlAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_PolyhdrlSurface ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE apolyhdrlsurfacearray ST_PolyhdrlSurface
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set apolyhdrlsurfacearray to an empty array.
      SET apolyhdrlsurfacearray = CAST(ARRAY[] AS
         ST_PolyhdrlSurface ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_PolyhdrlSurface value,
         then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_PolyhdrlSurface) THEN
            SIGNAL SQLSTATE '2FF69'
               SET MESSAGE_TEXT = 'element is not an ST_PolyhdrlSurface
               type';
         END IF;
         -- Cast the current element as an ST_PolyhdrlSurface and
         -- concatenate it to the end of apolyhdrlsurfacearray.
         SET apolyhdrlsurfacearray = apolyhdrlsurfacearray ||
            CAST(ageometryarray[counter] AS ST_PolyhdrlSurface);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_PolyhdrlSurface array
      RETURN apolyhdrlsurfacearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_PolyhdrlSurface ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToPolyhdrlAry
      (ST_PolyhdrlSurface ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToPolyhdrlAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToPolyhdrlAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_PolyhdrlSurface* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_PolyhdrlSurface type*.

      ii) Otherwise, return an *ST_PolyhdrlSurface* ARRAY value containing each element of *ageometryarray* cast as an *ST_PolyhdrlSurface* value.

3) Use the function *ST_ToPolyhdrlAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_PolyhdrlSurface* ARRAY value.

18.1.24 ST_ToTINAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_TIN valued elements to an ST_TIN ARRAY value.

Definition

```
CREATE FUNCTION ST_ToTINAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_TIN ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE atinarray ST_TIN ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set atinarray to an empty array.
      SET atinarray = CAST(ARRAY[] AS
         ST_TIN ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_TIN value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_TIN) THEN
            SIGNAL SQLSTATE '2FF70'
               SET MESSAGE_TEXT = 'element is not an ST_TIN type';
         END IF;
         -- Cast the current element as an ST_TIN and
         -- concatenate it to the end of atinarray.
         SET atinarray = atinarray ||
            CAST(ageometryarray[counter] AS ST_TIN);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_TIN array
      RETURN atinarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_TIN ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToTINAry
      (ST_TIN ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToTINAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToTINAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

i) If any element of *ageometryarray* is not an *ST_TIN* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_TIN type*.

ii) Otherwise, return an *ST_TIN* ARRAY value containing each element of *ageometryarray* cast as an *ST_TIN* value.

3) Use the function *ST_ToTINAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_TIN* ARRAY value.

18.1.25 ST_ToCompSurfAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_CompoundSurface valued elements to an ST_CompoundSurface ARRAY value.

Definition

```
CREATE FUNCTION ST_ToCompSurfAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_CompoundSurface ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE acompoundsurfacearray ST_CompoundSurface
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set acompoundsurfacearray to an empty array.
      SET acompoundsurfacearray = CAST(ARRAY[] AS
         ST_CompoundSurface ARRAY[ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_CompoundSurface value,
         then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_CompoundSurface) THEN
            SIGNAL SQLSTATE '2FF69'
               SET MESSAGE_TEXT = 'element is not an ST_CompoundSurface
               type';
         END IF;
         -- Cast the current element as an ST_CompoundSurface and
         -- concatenate it to the end of aCompoundsurfacearray.
         SET acompoundsurfacearray = acompoundsurfacearray ||
            CAST(ageometryarray[counter] AS ST_CompoundSurface);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_CompoundSurface array
      RETURN acompoundsurfacearray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_CompoundSurface ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCompSurfAry
      (ST_CompoundSurface ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToCompSurfAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToCompSurfAry(ST_Geometry ARRAY)*:

   a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

   b) Case:

      i) If any element of *ageometryarray* is not an *ST_CompoundSurface* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_CompoundSurface type*.

      ii) Otherwise, return an *ST_CompoundSurface* ARRAY value containing each element of *ageometryarray* cast as an *ST_CompoundSurface* value.

3) Use the function *ST_ToCompSurfAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_CompoundSurface*ARRAY value.

18.1.26 ST_ToBRepSolidAry Cast Function

Purpose

Cast an ST_Geometry ARRAY value that contains only ST_BRepSolid valued elements to an ST_BRepSolid ARRAY value.

Definition

```
CREATE FUNCTION ST_ToBRepSolidAry
   (ageometryarray ST_Geometry ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_BRepSolid ARRAY[ST_MaxGeometryArrayElements]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE counter INTEGER;
      DECLARE abrepsolidarray ST_BRepSolid
         ARRAY[ST_MaxGeometryArrayElements];

      -- If ageometryarray is the null value or contains null elements,
      -- then raise an exception.
      CALL ST_CheckNulls(ageometryarray);
      -- Set abrepsolidarray to an empty array.
      SET abrepsolidarray = CAST(ARRAY[] AS
         ST_BRepSolid [ST_MaxGeometryArrayElements]);
      SET counter = 1;
      -- For each element in ageometryarray
      WHILE counter <= CARDINALITY(ageometryarray) DO
         -- If the current element is not an ST_BRepSolid value, then
         -- raise an exception.
         IF ageometryarray[counter] IS NOT OF (ST_BRepSolid) THEN
            SIGNAL SQLSTATE '2FF69'
               SET MESSAGE_TEXT = 'element is not an ST_BRepSolid type';
         END IF;
         -- Cast the current element as an ST_BRepSolid and
         -- concatenate it to the end of abrepsolidarray.
         SET abrepsolidarray = abrepsolidarray ||
            CAST(ageometryarray[counter] AS ST_BRepSolid);
         SET counter = counter + 1;
      END WHILE;
      -- Return an ST_BRepSolid array
      RETURN abrepsolidarray;
   END

CREATE CAST(ST_Geometry ARRAY[ST_MaxGeometryArrayElements]
   AS ST_BRepSolid ARRAY[ST_MaxGeometryArrayElements])
   WITH FUNCTION ST_ToCompSurfAry
      (ST_BRepSolid ARRAY[ST_MaxGeometryArrayElements])
   AS ASSIGNMENT
```

Definitional Rules

1) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

Description

1) The function *ST_ToBRepSolidAry(ST_Geometry ARRAY)* takes the following input parameters:

   a) an *ST_Geometry* ARRAY value *ageometryarray*.

2) For the null-call function *ST_ToBRepSolidAry(ST_Geometry ARRAY)*:

a) Call the procedure *ST_CheckNulls(ST_Geometry ARRAY)* to check if *ageometryarray* is the null value or contains null elements.

b) Case:

i) If any element of *ageometryarray* is not an *ST_BRepSolid* value, then an exception condition is raised: *SQL/MM Spatial exception – element is not an ST_BRepSolid type*.

ii) Otherwise, return an *ST_BRepSolid* ARRAY value containing each element of *ageometryarray* cast as an *ST_BRepSolid* value.

3) Use the function *ST_ToBRepSolidAry(ST_Geometry ARRAY)* to define an implicitly invocable cast function to cast an *ST_Geometry* ARRAY value to an *ST_BRepSolid* ARRAY value.

19 SQL/MM Spatial Information Schema

19.1 Introduction

The SQL/MM Spatial Information Schema views are defined as being in a schema named *ST_INFORMTN_SCHEMA* enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views is granted to PUBLIC WITH GRANT OPTION so that they can be queried by any user and so that SELECT privilege can be further granted on views that reference these Information Schema views. How these views are updated is implementation-defined.

In order to provide access to the same information that is available via the *ST_INFORMTN_SCHEMA* to an SQL-Agent in an SQL-environment where the SQL-implementation does not support Feature F391, "Long identifiers" in Part 2 of ISO/IEC 9075, alternative views are provided that use only short identifiers.

An implementation may define objects that are associated with *ST_INFORMTN_SCHEMA* that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

19.2 ST_GEOMETRY_COLUMNS view

Purpose

Identify the columns in any table that have ST_Geometry or one of its subtypes as its declared type.

Definition

```
CREATE VIEW ST_GEOMETRY_COLUMNS AS
   WITH RECURSIVE TYPES ( TYPE_CATALOG, TYPE_SCHEMA, TYPE_NAME ) AS
       ( VALUES ( ST_TypeCatalogName, ST_TypeSchemaName, 'ST_GEOMETRY' )
         UNION ALL
         SELECT h.USER_DEFINED_TYPE_CATALOG, h.USER_DEFINED_TYPE_SCHEMA,
                h.USER_DEFINED_TYPE_NAME
           FROM INFORMATION_SCHEMA.DIRECT_SUPERTYPES AS h
                  JOIN
                TYPES AS t ON
                  ( h.SUPERTYPE_CATALOG = t.TYPE_CATALOG AND
                    h.SUPERTYPE_SCHEMA  = t.TYPE_SCHEMA AND
                    h.SUPERTYPE_NAME    = t.TYPE_NAME )
       )
       ( SELECT c.TABLE_CATALOG, c.TABLE_SCHEMA,
            c.TABLE_NAME, c.COLUMN_NAME, g.SRS_NAME,
            ( SELECT s.SRS_ID
                FROM ST_DEFINITION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS
                        AS s
               WHERE s.SRS_NAME = g.SRS_NAME
            ) AS SRS_ID
           FROM INFORMATION_SCHEMA.COLUMNS AS c
                  LEFT OUTER JOIN
                ST_DEFINITION_SCHEMA.ST_GEOMETRY_COLUMNS AS g ON
                     ( c.TABLE_CATALOG = g.TABLE_CATALOG AND
                       c.TABLE_SCHEMA  = g.TABLE_SCHEMA AND
                       c.TABLE_NAME    = g.TABLE_NAME AND
                       c.COLUMN_NAME   = g.COLUMN_NAME )
          WHERE ( c.UDT_CATALOG, c.UDT_SCHEMA, c.UDT_NAME ) IN
            ( SELECT TYPE_CATALOG, TYPE_SCHEMA, TYPE_NAME FROM TYPES ) )
```

Definitional Rules

1) *ST_TypeCatalogName* is the implementation-defined character representation of the name of the catalog, which contains the descriptor of the data type *ST_Geometry*.

2) *ST_TypeSchemaName* is the implementation-defined character representation of the name of the schema, which contains the descriptor of the data type *ST_Geometry*.

19.3 ST_SPATIAL_REFERENCE_SYSTEMS view

Purpose

List the supported spatial reference systems.

Definition

```
CREATE VIEW ST_SPATIAL_REFERENCE_SYSTEMS AS
   SELECT SRS_NAME, SRS_ID,
         ORGANIZATION, ORGANIZATION_COORDSYS_ID,
         DEFINITION, DESCRIPTION
      FROM ST_DEFINITION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS
```

19.4 ST_UNITS_OF_MEASURE view

Purpose

List the supported units of measure.

Definition

```
CREATE VIEW ST_UNITS_OF_MEASURE AS
   SELECT UNIT_NAME, UNIT_TYPE, CONVERSION_FACTOR, DESCRIPTION
      FROM ST_DEFINITION_SCHEMA.ST_UNITS_OF_MEASURE
```

19.5 ST_SIZINGS view

Purpose

List the implementation-defined meta-variables and their values.

Definition

```
CREATE VIEW ST_SIZINGS AS
   SELECT VARIABLE_NAME, SUPPORTED_VALUE, DESCRIPTION
      FROM ST_DEFINITION_SCHEMA.ST_SIZINGS
```

19.6 Short name views

Purpose

Provide alternative views that use only identifiers that do not require Feature F391, "Long identifiers", in Part 2 of ISO/IEC 9075.

Definition

```
CREATE VIEW GEOMETRY_COLUMNS AS
   SELECT
         TABLE_CATALOG AS F_TABLE_CATALOG,
         TABLE_SCHEMA AS F_TABLE_SCHEMA,
         TABLE_NAME AS F_TABLE_NAME,
         COLUMN_NAME AS F_GEOMETRY_COLUMN,
         SRS_NAME,
         SRS_ID AS SRID
      FROM ST_INFORMTN_SCHEMA.ST_GEOMETRY_COLUMNS

CREATE VIEW SPATIAL_REF_SYS AS
   SELECT
         SRS_NAME,
         SRS_ID AS SRID,
         ORGANIZATION AS AUTH_NAME,
         ORGANIZATION_COORDSYS_ID AS AUTH_ID,
         DEFINITION AS SRTEXT
      FROM ST_DEFINITION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS

CREATE VIEW ST_UNITS AS
   SELECT UNIT_NAME, UNIT_TYPE, CONVERSION_FACTOR, DESCRIPTION
      FROM ST_DEFINITION_SCHEMA.ST_UNITS_OF_MEASURE
```

20 SQL/MM Spatial Definition Schema

20.1 Introduction

The only purpose of the SQL/MM Spatial Definition Schema is to provide a data model to support the *ST_INFORMTN_SCHEMA* and to assist understanding. The base tables of the SQL/MM Spatial Definition Schema are defined as being in a schema named *ST_DEFINITION_SCHEMA*. The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1. The function of the definition is stated.
2. The SQL definition of the object is presented as a <table definition>.
3. An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

20.2 ST_GEOMETRY_COLUMNS base table

Purpose

List the columns in any table that have ST_Geometry or one of its subtypes as declared type and their associated spatial reference systems.

Definition

```
CREATE TABLE ST_GEOMETRY_COLUMNS
   (
   TABLE_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   TABLE_SCHEMA INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   TABLE_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   COLUMN_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   SRS_NAME CHARACTER VARYING(ST_MaxSRSNameLength),

   CONSTRAINT ST_GEOMETRY_COLUMNS_PRIMARY_KEY
      PRIMARY KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME),
   CONSTRAINT SRS_SUPPORTED FOREIGN KEY(SRS_NAME)
      REFERENCES ST_SPATIAL_REFERENCE_SYSTEMS(SRS_NAME),
   CONSTRAINT COLUMN_EXISTS
      FOREIGN KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
      REFERENCES INFORMATION_SCHEMA.COLUMNS
         (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
   )
```

Definitional Rules

1) *ST_MaxSRSNameLength* is the implementation-defined maximum length used for the character representation of the identifier of a spatial reference system.

Description

1) The values of *TABLE_CATALOG*, *TABLE_SCHEMA*, and *TABLE_NAME* are the catalog name, the unqualified schema name, and the qualified identifier, respectively, of the table containing the column being described.
2) The values of *COLUMN_NAME* are the names of the columns being described. The column shall have a declared type of *ST_Geometry* or one of its subtypes.
3) The values of *SRS_NAME* are the names of the spatial reference systems associated with each column. If no spatial reference system is associated with the column, *SRS_NAME* represents the null value.

20.3 ST_SPATIAL_REFERENCE_SYSTEMS base table

Purpose

List the supported spatial reference systems.

Definition

```
CREATE TABLE ST_SPATIAL_REFERENCE_SYSTEMS
   (
   SRS_NAME CHARACTER VARYING(ST_MaxSRSNameLength) NOT NULL,
   SRS_ID INTEGER NOT NULL,
   ORGANIZATION CHARACTER VARYING(ST_MaxOrganizationNameLength),
   ORGANIZATION_COORDSYS_ID INTEGER,
   DEFINITION CHARACTER VARYING(ST_MaxSRSDefinitionLength) NOT NULL,
   DESCRIPTION CHARACTER VARYING(ST_MaxDescriptionLength),

   CONSTRAINT ST_SRS_NAME_PRIMARY_KEY PRIMARY KEY(SRS_NAME),
   CONSTRAINT SRS_ID_UNIQUE UNIQUE (SRS_ID),
   CONSTRAINT ORGANIZATION_NULL
      CHECK (
         ( ORGANIZATION IS NULL AND
           ORGANIZATION_COORDSYS_ID IS NULL ) OR
         ( ORGANIZATION IS NOT NULL AND
           ORGANIZATION_COORDSYS_ID IS NOT NULL ) ),
   CONSTRAINT ORGANIZATION_UNIQUE
      CHECK (
         ( ORGANIZATION IS NULL AND
           ORGANIZATION_COORDSYS_ID IS NULL ) OR
         ( 1 = ( SELECT COUNT(*)
            FROM ST_SPATIAL_REFERENCE_SYSTEMS AS t
            WHERE t.ORGANIZATION = ORGANIZATION AND
               t.ORGANIZATION_COORDSYS_ID = ORGANIZATION_COORDSYS_ID ) ) )
   )
```

Definitional Rules

1) *ST_MaxSRSNameLength* is the implementation-defined maximum length used for the character representation of the identifier of a spatial reference system.
2) *ST_MaxOrganizationNameLength* is the implementation-defined maximum length used for the character representation of an organization name.
3) *ST_MaxSRSDefinitionLength* is the implementation-defined maximum length for the well-known text representation of a spatial reference system.
4) *ST_MaxDescriptionLength* is the implementation-defined maximum length used for the character representation of a description.

Description

1) The values of *SRS_NAME* are the names of the spatial reference systems.
2) The values of *SRS_ID* are numerical identifiers of spatial reference systems.
3) The values of *ORGANIZATION* are character representations of the name of the organization that defined the spatial reference system.
4) The values of *ORGANIZATION_COORDSYS_ID* are numerical identifiers for the spatial reference system as assigned by the organization represented in the *ORGANIZATION* column.
5) The values of *DEFINITION* are the character representations of the well-known text representations <spatial reference system> of a spatial reference system.

6) The values of *DESCRIPTION* are character representations of the description of the spatial reference systems.

NOTE The BNF for <spatial reference system> is defined in Subclause 13.1.2, "ST_SpatialRefSys Methods".

20.4 ST_UNITS_OF_MEASURE base table

Purpose

List the supported units of measure.

Definition

```
CREATE TABLE ST_UNITS_OF_MEASURE
   (
   UNIT_NAME CHARACTER VARYING(ST_MaxUnitNameLength) NOT NULL,
   UNIT_TYPE CHARACTER VARYING(ST_MaxUnitTypeLength) NOT NULL,
   CONVERSION_FACTOR DOUBLE PRECISION NOT NULL,
   DESCRIPTION CHARACTER VARYING(ST_MaxDescriptionLength),

   CONSTRAINT ST_UNITS_PRIMARY_KEY PRIMARY KEY ( UNIT_NAME ),
   CONSTRAINT UNIT_TYPE_VALUE
      CHECK ( UNIT_TYPE IN ( 'ANGULAR', 'LINEAR' ) ),
   CONSTRAINT FACTOR_VALUE
      CHECK ( CONVERSION_FACTOR > 0.0 )
   )
```

Definitional Rules

1) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

2) *ST_MaxUnitTypeLength* is the implementation-defined maximum length used for the character representation of the type of a unit of measure.

3) *ST_MaxDescriptionLength* is the implementation-defined maximum length used for the character representation of a description.

Description

1) The values of *UNIT_NAME* are character representations of the identifiers of units of measure supported by an implementation.

2) The values of *UNIT_TYPE* are character representations of the type of units of measure supported by an implementation. The type of a unit of measure can either be 'ANGULAR' or 'LINEAR'.

3) The values of *CONVERSION_FACTOR* are the factors to convert a value in the specific unit to a value in the base unit. The base unit is that unit with the same *UNIT_TYPE* value and with a *CONVERSION_FACTOR* of 1 (one). For linear units, the base unit is 'METRE'. For angular units, the base unit is 'RADIAN'.

4) The values of *DESCRIPTION* are character representations of the description of the units of measure.

20.5 ST_SIZINGS base table

Purpose

List the implementation-defined meta-variables and their values.

Definition

```
CREATE TABLE ST_SIZINGS
   (
   VARIABLE_NAME CHARACTER VARYING(ST_MaxVariableNameLength) NOT NULL,
   SUPPORTED_VALUE INTEGER,
   DESCRIPTION CHARACTER VARYING(ST_MaxDescriptionLength),

   CONSTRAINT ST_SIZINGS_PRIMARY_KEY PRIMARY KEY ( VARIABLE_NAME )
```

)

Definitional Rules

1) *ST_MaxVariableNameLength* is the implementation-defined maximum length used for the character representation of an implementation-defined meta-variable.
2) *ST_MaxDescriptionLength* is the implementation-defined maximum length used for the character representation of a description.

Description

1) The values of *VARIABLE_NAME* are character representations of the identifiers of the implementation-defined meta-variables.
2) The values of *SUPPORTED_VALUE* are:
   a) 0 (zero): The implementation either places no limit on this implementation-defined meta-variable or the implementation cannot determine the limit.
   b) the null value: The implementation does not support any features for which this implementation-defined meta-variable is applicable.
   c) Any other value: The maximum size supported by the implementation for this implementation-defined meta-variable.
3) The values of *DESCRIPTION* are character representations of the description of the implementation-defined meta-variables.

21 SQL/MM Linear Referencing Information and Definition Schemas

21.1 Information Schema

21.1.1 Introduction

The views *ST_LR_COLUMNS* and *ST_LRMS* are defined as being in a schema named *ST_INFORMTN_SCHEMA* enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views is granted to PUBLIC WITH GRANT OPTION so that they can be queried by any user and so that SELECT privilege can be further granted on views that reference these Information Schema views. How these views are updated is implementation-defined.

An implementation may define objects that are associated with *ST_INFORMTN_SCHEMA* that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

21.1.2 ST_LR_COLUMNS view

Purpose

Identify the columns in any table that have any linearly referenced user defined type as a declared type and their associated linear referencing methods.

Definition

```
CREATE VIEW ST_LR_COLUMNS AS
   SELECT
         TABLE_CATALOG,
         TABLE_SCHEMA,
         TABLE_NAME,
         COLUMN_NAME,
         LRM_ID AS LRMID
      FROM ST_DEFN_SCHEMA.ST_LR_COLUMNS
```

21.1.3 ST_LRMS view

Purpose

List the supported linear referencing methods.

Definition

```
CREATE VIEW ST_LRMS AS
   SELECT
         LRM_NAME,
         LRM_ID AS LRMID,
         ORGANIZATION AS AUTH_NAME,
         ORGANIZATION_COORDSYS_ID AS AUTH_ID,
         DEFINITION AS WKT,
         DESCRIPTION,
         LRM
      FROM ST_DEFN_SCHEMA.ST_LRMS
```

21.2 Definition Schemata

21.2.1 Introduction

The only purpose of the SQL/MM Linear Referencing Schemata is to provide a data model to support the *ST_INFORMTN_SCHEMA* and to assist understanding. The base tables *ST_LR_COLUMNS* and *ST_LRMS* are defined as being in a schema named *ST_DEFN_SCHEMA*.

The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1. The function of the definition is stated.
2. The SQL definition of the object is presented as a <table definition>.
3. An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

21.2.2 ST_LR_COLUMNS base table

Purpose

List the columns in any table that have any linearly referenced user defined type as a declared type and their associated linear referencing methods.

Definition

```
CREATE TABLE ST_LR_COLUMNS
   (
   TABLE_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   TABLE_SCHEMA INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   TABLE_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   COLUMN_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
   LRM_ID INTEGER,

   CONSTRAINT ST_LR_COLUMNS_PRIMARY_KEY
      PRIMARY KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME),
   CONSTRAINT LRM_SUPPORTED FOREIGN KEY(LRM_ID)
      REFERENCES ST_LINEAR_REFERENCING_METHODS(LRM_ID),
   CONSTRAINT COLUMN_EXISTS
      FOREIGN KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
      REFERENCES INFORMATION_SCHEMA.COLUMNS
         (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
   )
```

Description

1) The values of *TABLE_CATALOG*, *TABLE_SCHEMA*, and *TABLE_NAME* are the catalog name, the unqualified schema name, and the qualified identifier, respectively, of the table containing the column being described.
2) The values of *COLUMN_NAME* are the names of the columns being described. The column shall have a declared type of any linearly referenced user defined type.
3) The values of *LRM_ID* are the identifiers of the linear referencing methods associated with each column. If no linear referencing method is associated with the column, *LRM_ID* represents the null value.

21.2.3 ST_LRMS base table

Purpose

List the supported linear referencing methods.

Definition

```
CREATE TABLE ST_LRMS
   (
   LRM_NAME CHARACTER VARYING(ST_MaxLRMNameLength) NOT NULL,
   LRM_ID INTEGER NOT NULL,
   ORGANIZATION CHARACTER VARYING(ST_MaxOrganizationNameLength),
   ORGANIZATION_LRM_ID INTEGER,
   DEFINITION CHARACTER VARYING(ST_MaxLRAsText) NOT NULL,
   DESCRIPTION CHARACTER VARYING(ST_MaxDescriptionLength),
   LRM ST_LRM,

   CONSTRAINT LRM_ID_PRIMARY_KEY PRIMARY KEY(LRM_ID),
   CONSTRAINT LRM_NAME_UNIQUE UNIQUE (LRM_NAME),
   CONSTRAINT ORGANIZATION_NULL
      CHECK (
         ( ORGANIZATION IS NULL AND
          ORGANIZATION_LRM_ID IS NULL ) OR
         ( ORGANIZATION IS NOT NULL AND
          ORGANIZATION_LRM_ID IS NOT NULL ) ),
   CONSTRAINT ORGANIZATION_UNIQUE
      CHECK (
         ( ORGANIZATION IS NULL AND
          ORGANIZATION_LRM_ID IS NULL ) OR
         ( 1 = ( SELECT COUNT(*)
            FROM ST_LRMS AS t
            WHERE t.ORGANIZATION = ORGANIZATION AND
               t.ORGANIZATION_LRM_ID = ORGANIZATION_LRM_ID ) ) )
   )
```

Definitional Rules

1) *ST_MaxLRMNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a linear referencing method.
2) *ST_MaxOrganizationNameLength* is the implementation-defined maximum length used for the character representation of an organization name.
3) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.
4) *ST_MaxDescriptionLength* is the implementation-defined maximum length used for the character representation of a description.

Description

1) The values of *LRM_NAME* are the names of the linear referencing methods.
2) The values of *LRM_ID* are numerical identifiers of linear referencing methods.
3) The values of *ORGANIZATION* are character representations of the name of the organization that defined the linear referencing methods.
4) The values of *ORGANIZATION_LRM_ID* are numerical identifiers for the linear referencing method as assigned by the organization represented in the *ORGANIZATION* column.
5) The values of *DEFINITION* are the character representations of the well-known text representations <lrm representation> of a linear referencing method.
6) The values of *DESCRIPTION* are character representations of the description of the linear referencing methods.

7) The values of *LRM* are values of the *ST_LRM* Linear Referencing Method user-defined type.

8) For each row in *ST_LRMS*, if *theLRMID* is the INTEGER *LRM_ID* value and *theLRM* is the *ST_LRM* *LRM* value, then if *theLRM* is not NULL, then *theLRMID* shall be equal to *theLRM.ST_LRMID().*

9) At least one of *DEFINITION* or *LRM* shall not be NULL. If both are not NULL, then *DEFINITION* shall be the well-known text representation of *LRM*.

NOTE The BNF for <lrm representation> is defined in Subclause 15.14.3, "<lrm text representation>".

22 Status Codes

The character string value returned in an SQLSTATE parameter comprises a 2-character class value followed by a 3-character subclass value. The class value for each condition and the subclass value or values for each class value are specified in Table 16 — SQLSTATE class and subclass values.

The "Category" column has the following meanings: "S" means that the class value given corresponds to successful completion and is a completion condition; "W" means that the class value given corresponds to a successful completion but with a warning and is a completion condition; "N" means that the class value corresponds to a no-data situation and is a completion condition; "X" means that the class value given corresponds to an exception condition.

For a successful completion code but with a warning, the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class value for *warning* (defined in Subclause 23.1, "SQLSTATE" in Part 2 of ISO/IEC 9075).

For an exception completion code, the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class value *SQL routine exception* (defined in Subclause 23.1, "SQLSTATE" in Part 2 of ISO/IEC 9075-2).

Table 16 — SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
W	SQL/MM Spatial warning	01	invalid position	F01
X	SQL/MM Spatial exception	2F	invalid argument	F02
X	SQL/MM Spatial exception	2F	null argument	F03
X	SQL/MM Spatial exception	2F	invalid intersection matrix	F04
X	SQL/MM Spatial exception	2F	duplicate value	F05
X	SQL/MM Spatial exception	2F	element is an empty set	F06
X	SQL/MM Spatial exception	2F	null exterior ring	F07
X	SQL/MM Spatial exception	2F	element is not a valid type	F08
X	SQL/MM Spatial exception	2F	element is a null value	F09
X	SQL/MM Spatial exception	2F	mixed spatial reference systems	F10
X	SQL/MM Spatial exception	2F	non-contiguous curves	F11
X	SQL/MM Spatial exception	2F	curve value is not a linestring value	F12
X	SQL/MM Spatial exception	2F	attempted division by zero	F13
X	SQL/MM Spatial exception	2F	unsupported unit specified	F14
X	SQL/MM Spatial exception	2F	failed to transform geometry	F15
X	SQL/MM Spatial exception	2F	not an empty set	F16
X	SQL/MM Spatial exception	2F	empty point value	F17
X	SQL/MM Spatial exception	2F	point value not well formed	F18
X	SQL/MM Spatial exception	2F	points are equal	F19
X	SQL/MM Spatial exception	2F	linestring is not a line	F20
X	SQL/MM Spatial exception	2F	degenerate line has no direction	F21
X	SQL/MM Spatial exception	2F	invalid well-known text representation	F22
X	SQL/MM Spatial exception	2F	invalid well-known binary representation	F23
X	SQL/MM Spatial exception	2F	invalid GML representation	F24
X	SQL/MM Spatial exception	2F	mixed coordinate dimensions	F25
W	SQL/MM Spatial warning	01	disconnected points not included in result	F26
X	SQL/MM Spatial exception	2F	coincident edge	F27
X	SQL/MM Spatial exception	2F	coincident node	F28
X	SQL/MM Spatial exception	2F	curve not simple	F29
X	SQL/MM Spatial exception	2F	edge crosses node	F30
X	SQL/MM Spatial exception	2F	empty network	F31
X	SQL/MM Spatial exception	2F	empty topology	F32
X	SQL/MM Spatial exception	2F	end node not geometry end point	F33
X	SQL/MM Spatial exception	2F	geometry crosses a node	F34
X	SQL/MM Spatial exception	2F	geometry crosses an edge	F35

Category	Condition	Class	Subcondition	Subclass
X	SQL/MM Spatial exception	2F	geometry intersects an edge	F36
X	SQL/MM Spatial exception	2F	geometry not within face	F37
X	SQL/MM Spatial exception	2F	link has null geometry	F38
X	SQL/MM Spatial exception	2F	nodes in different faces	F39
X	SQL/MM Spatial exception	2F	non-connected edges	F40
X	SQL/MM Spatial exception	2F	non-connected links	F41
X	SQL/MM Spatial exception	2F	non-empty view	F42
X	SQL/MM Spatial exception	2F	non-existent edge	F43
X	SQL/MM Spatial exception	2F	non-existent face	F44
X	SQL/MM Spatial exception	2F	non-existent link	F45
X	SQL/MM Spatial exception	2F	non-existent node	F46
X	SQL/MM Spatial exception	2F	non-existent schema	F47
X	SQL/MM Spatial exception	2F	non-existent view	F48
X	SQL/MM Spatial exception	2F	not a logical link	F49
X	SQL/MM Spatial exception	2F	not isolated node	F50
X	SQL/MM Spatial exception	2F	not within face	F51
X	SQL/MM Spatial exception	2F	other edges connected	F52
X	SQL/MM Spatial exception	2F	other links connected	F53
X	SQL/MM Spatial exception	2F	point not on edge	F54
X	SQL/MM Spatial exception	2F	point not on link	F55
X	SQL/MM Spatial exception	2F	schema already exists	F56
X	SQL/MM Spatial exception	2F	start node not geometry start point	F57
X	SQL/MM Spatial exception	2F	null node geometry	F58
X	SQL/MM Spatial exception	2F	unknown spatial reference system	F59
X	SQL/MM Spatial exception	2F	universal face has no geometry	F60
X	SQL/MM Spatial exception	2F	invalid universal face	F61
X	SQL/MM Spatial exception	2F	invalid topology name	F62
X	SQL/MM Spatial exception	2F	topology privilege denied	F63
X	SQL/MM Spatial exception	2F	invalid network name	F64
X	SQL/MM Spatial exception	2F	network privilege denied	F65
X	SQL/MM Spatial exception	2F	triangles cannot have holes	F66
X	SQL/MM Spatial exception	2F	polygon value is not a triangle value	F67
X	SQL/MM Spatial exception	2F	element is not an ST_Triangle type	F68
X	SQL/MM Spatial exception	2F	element is not an ST_PolyhdrlSurface type	F69
X	SQL/MM Spatial exception	2F	element is not an ST_TIN type	F70
X	SQL/MM Spatial exception	2F	at least 3 points are required	F71
X	SQL/MM Spatial exception	2F	both geometries must be 3D	F72
X	SQL/MM Spatial exception	2F	geometry is not 3D	F73
X	SQL/MM Spatial exception	2F	invalid geometry	F74
X	SQL/MM Spatial exception	2F	exterior ring must have exactly 4 points	F75
X	SQL/MM Spatial exception	2F	curve has multiple segments	F76
X	SQL/MM Spatial exception	2F	exactly three points are required	F77
X	SQL/MM Spatial exception	2F	points are collinear	F78
X	SQL/MM Spatial exception	2F	the given distance is longer than curve	F79
X	SQL/MM Spatial exception	2F	the point is not on the curve	F80
X	SQL/MM Spatial exception	2F	invalid LRM	F81
W	SQL/MM Spatial warning	01	changing default measure may invalidate position expressions using this linear element	F82
X	SQL/MM Spatial exception	2F	potentially incompatible referent position and location	F83
X	SQL/MM Spatial exception	2F	missing measure value(s)	F84
X	SQL/MM Spatial exception	2F	non-contiguous surfaces	F85

Category	Condition	Class	Subcondition	Subclass
X	SQL/MM Spatial exception	2F	incorrect number of vectors	F86
X	SQL/MM Spatial exception	2F	cannot translate	F87
X	SQL/MM Spatial exception	2F	indeterminate equality	F88
X	SQL/MM Spatial exception	2F	offset unit must be specified	F89
X	SQL/MM Spatial exception	2F	towards referent requires a from referent	F90
X	SQL/MM Spatial exception	2F	illegal with vector offset	F91
X	SQL/MM Spatial exception	2F	illegal with lateral offset	F92
X	SQL/MM Spatial exception	2F	illegal with vertical offset	F93
X	SQL/MM Spatial exception	2F	illegal with offset referent description	F94
X	SQL/MM Spatial exception	2F	illegal with offset referent geometry	F95
X	SQL/MM Spatial exception	2F	mixed Is3D	F96
X	SQL/MM Spatial exception	2F	m coordinates not allowed	F97
X	SQL/MM Spatial exception	2F	null exterior shell	F98

23 Conformance

23.1 Requirements for conformance

A conforming implementation shall support one of the mandatory groups of public user-defined types and routines given by this part of ISO/IEC 13249 and may in addition support some or all of the optional user-defined types and routines.

A conforming implementation shall support the views comprising the Spatial Information Schema as defined in Clause 18, "SQL/MM Spatial Information Schema".

23.2 Features of ISO/IEC 9075 required for this part of ISO/IEC 13249

1) This part of ISO/IEC 13249 requires the following features defined in ISO/IEC 9075 for the mandatory groups of public user-defined types and routines:

   - Feature S024, "Enhanced structured types"

   - Feature S241, "Transform functions"

   - Feature T322, "Overloading of SQL-invoked functions and procedures"

2) This part of ISO/IEC 13249 requires the following features defined in ISO/IEC 9075 for the optional user-defined routines that have ARRAY data types defined for parameters or return values:

   - Feature S092, "Arrays of user-defined types"

   - Feature S201, "SQL-invoked routines on arrays"

3) This part of ISO/IEC 13249 requires the following feature defined in ISO/IEC 9075 for the optional user-defined routines implemented as external SQL-invoked functions that have ARRAY data types defined for parameters or return values:

   - Feature T571, "Array-returning external SQL-invoked functions"

4) This part of ISO/IEC 13249 requires the following features defined in ISO/IEC 9075 for the optional user-defined routines that have MULTISET data types defined for parameters or return values:

   - Feature S272, "Multisets of user-defined types"

   - Feature S202, "SQL-invoked routines on multisets"

5) This part of ISO/IEC 13249 requires the following feature defined in ISO/IEC 9075 for the optional user-defined routines implemented as external SQL-invoked functions that have ARRAY data types defined for parameters or return values:

   - Feature T572, "Multiset-returning external SQL-invoked functions"

6) The ST_InitTopoGeo and ST_InitTopoNet procedures in this part of ISO/IEC 13249 requires either of the following features defined in ISO/IEC 9075:

   - Feature T651, " SQL-schema statements in SQL routines"

   - Feature T653, "SQL-schema statements in external routines"

   However, a conforming implementation can choose alternate means to address the contained functionality without having to use these procedures.

23.3 Claims of conformance

Claims of conformance to this part of ISO/IEC 13249 shall state:

1) Which of the following mandatory groups of public user-defined types and routines are supported:

   a) ST_Point, ST_LineString, ST_Polygon, and ST_GeomCollection with non-instantiable types ST_Geometry, ST_Curve, and ST_Surface.

   b) ST_Point, ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon, and ST_GeomCollection with non-instantiable types ST_Geometry, ST_Curve, ST_Surface, ST_MultiCurve, and ST_MultiSurface.

2) Whether or not methods with <collection type>s are supported in <SQL parameter declaration> and <returns clause> of a <SQL-invoked routine> and if so then the following methods shall be supported for the following data types:

   a) For the ST_Geometry type (Subclause 5.1, "ST_Geometry Type and Routines"):

      i) The method ST_EnvelopeAsPts() (Subclause 5.1.20, "ST_EnvelopeAsPts Method").

   b) For the ST_LineString type (Subclause 7.2, "ST_LineString Type and Routines"):

      i) The method ST_LineString(ST_Point ARRAY) and the method ST_LineString(ST_Point ARRAY, INTEGER) (Subclause 7.2.2, "ST_LineString Methods").

      ii) The ST_Points methods (Subclause 7.2.3, "ST_Points Methods").

      iii) The ST_ToLineString method (Subclause 5.1.55, "Cast").

      iv) The CAST ST_Geometry AS ST_LineString (Subclause 5.1.55, "Cast").

   c) If the ST_CircularString type (Subclause 7.3, "ST_CircularString Type and Routines") is supported:

      i) The method ST_CircularString(ST_Point ARRAY) and the method ST_CircularString(ST_Point ARRAY, INTEGER) (Subclause 7.3.2, "ST_CircularString Methods").

      ii) The ST_Points methods (Subclause 7.3.3, "ST_Points Methods").

      iii) The ST_ToCircular method (Subclause 5.1.55, "Cast").

      iv) The CAST ST_Geometry AS ST_CircularString (Subclause 5.1.55, "Cast").

   d) If the ST_Circle type (Subclause 7.4, "ST_Circle Type and Routines") is supported:

      i) The method ST_Circle(ST_Point ARRAY) and the method ST_Circle(ST_Point ARRAY, INTEGER) (Subclause 7.4.2, "ST_Circle Methods").

      ii) The ST_Points methods (Subclause 7.4.3, "ST_Points Methods").

      iii) The ST_ToCircle method (Subclause 5.1.55, "Cast").

      iv) The CAST ST_Geometry AS ST_Circle (Subclause 5.1.55, "Cast").

   e) If the ST_GeodesicString type (Subclause 7.5, "ST_GeodesicString Type and Routines") is supported:

      i) The method ST_GeodesicString(ST_Point ARRAY) and the method ST_GeodesicString(ST_Point ARRAY, INTEGER) (Subclause 7.5.2, "ST_GeodesicString Methods").

      ii) The ST_Points methods (Subclause 7.5.3, "ST_Points Methods").

      iii) The ST_ToGeodesicString method (Subclause 5.1.55, "Cast").

      iv) The CAST ST_Geometry AS ST_GeodesicString (Subclause 5.1.55, "Cast").

   f) If the ST_EllipticalCurve type (Subclause 7.6, "ST_EllipticalCurve Type and Routines") is supported:

      i) The ST_ToElliptical method (Subclause 5.1.55, "Cast").

      ii) The CAST ST_Geometry AS ST_EllipticalCurve (Subclause 5.1.55, "Cast").

   g) If the ST_NURBSCurve type (Subclause 7.7, "ST_NURBSCurve Type and Routines") is supported:

      i) The method ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY), the method ST_NURBSCurve (INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, INTEGER), the method ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION) and the method ST_NURBSCurve(INTEGER, ST_NURBSPoint ARRAY, ST_Knot ARRAY, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER) (Subclause 7.7.2, "ST_NURBSCurve Methods").

   ii) The ST_ControlPoints methods (Subclause 7.7.4, "ST_ControlPoints Methods").
   iii) The ST_Knots methods (Subclause 7.7.5, "ST_Knots Methods").
   iv) The ST_ToNURBS method (Subclause 5.1.55, "Cast").
   v) The CAST ST_Geometry AS ST_NURBSCurve (Subclause 5.1.55, "Cast").

h) If the ST_Clothoid type (Subclause 7.8, "ST_Clothoid Type and Routines") is supported:
   i) The ST_ToClothoid method (Subclause 5.1.55, "Cast").
   ii) The CAST ST_Geometry AS ST_Clothoid (Subclause 5.1.55, "Cast").

i) If the ST_SpiralCurve type (Subclause 7.9, "ST_SpiralCurve Type and Routines") is supported:
   i) The ST_ToSpiral method (Subclause 5.1.55, "Cast").
   ii) The CAST ST_Geometry AS ST_SpiralCurve (Subclause 5.1.55, "Cast").

j) If the ST_CompoundCurve type (Subclause 7.10, "ST_CompoundCurve Type and Routines") is supported:
   i) The method ST_CompoundCurve(ST_Curve ARRAY) and method ST_CompoundCurve(ST_Curve ARRAY, INTEGER) (Subclause 7.10.2, "ST_CompoundCurve Methods").
   ii) The ST_Curves methods (Subclause 7.10.3, "ST_Curves Methods").
   iii) The ST_ToCompound method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_CompoundCurve (Subclause 5.1.55, "Cast").

k) If the ST_CurvePolygon type (Subclause 8.2, "ST_CurvePolygon Type and Routines") is instantiable:
   i) The method ST_CurvePolygon(ST_Curve, ST_Curve ARRAY) and the method ST_CurvePolygon(ST_Curve, ST_Curve ARRAY, INTEGER) (Subclause 8.2.2, "ST_CurvePolygon Methods").
   ii) The ST_InteriorRings methods (Subclause 8.2.4, "ST_InteriorRings Methods").
   iii) The ST_ToCurvePoly method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_CurvePolygon (Subclause 5.1.55, "Cast").

l) For the ST_Polygon type (Subclause 8.3, "ST_Polygon Type and Routines"):
   i) The method ST_Polygon(ST_LineString, ST_LineString ARRAY) and the method ST_Polygon(ST_LineString, ST_LineString ARRAY, INTEGER) (Subclause 8.3.2, "ST_Polygon Methods").
   ii) The ST_InteriorRings methods (Subclause 8.2.4, "ST_InteriorRings Methods").

m) If the ST_Triangle type (Subclause 8.4, "ST_Triangle Type and Routines"):
   i) The method ST_Triangle(ST_LineString, ST_LineString ARRAY) and the method ST_Triangle(ST_LineString, ST_LineString ARRAY, INTEGER) (Subclause 8.4.2, "ST_Triangle Methods").
   ii) The ST_InteriorRings methods (Subclause 8.4.6, "ST_InteriorRings Methods").
   iii) The ST_ToTriangle method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_Triangle (Subclause 5.1.55, "Cast").

n) If the ST_PolyhdrlSurface type (Subclause 8.5, "ST_PolyhdrlSurface Type and Routines"):
   i) The method ST_PolyhdrlSurface(ST_Polygon ARRAY) and the method ST_PolyhdrlSurface(ST_Polygon ARRAY, INTEGER) (Subclause 8.5.2, "ST_PolyhdrlSurface Methods").
   ii) The ST_Patches methods (Subclause 8.5.3, "ST_Patches Methods").
   iii) The ST_ToPolyhdrlSurf method (Subclause 5.1.55, "Cast").

iv) The CAST ST_Geometry AS ST_PolyhdrlSurface (Subclause 5.1.55, "Cast").

o) If the ST_TIN type (Subclause 8.6, "ST_TIN Type and Routines"):

i) The method ST_TIN(ST_Triangle ARRAY, ST_Point ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, DOUBLE PRECISION), the method ST_TIN(ST_Triangle ARRAY, ST_Point ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, DOUBLE PRECISION, INTEGER), the method ST_TIN(ST_Point ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, DOUBLE PRECISION), and the method ST_TIN(ST_Point ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, ST_LineString ARRAY, DOUBLE PRECISION, INTEGER), (Subclause 8.6.2, "ST_TIN Methods").

ii) The ST_TINElements methods (Subclause 8.6.3, "ST_TINElements Methods").

iii) The ST_TINTable methods (Subclause 8.6.5, "ST_TINTable Methods").

iv) The ST_Patches methods (Subclause 8.6.7, "ST_Patches Methods").

v) The ST_ToTIN method (Subclause 5.1.55, "Cast").

vi) The CAST ST_Geometry AS ST_TIN (Subclause 5.1.55, "Cast").

p) If the ST_CompoundSurface type (Subclause 8.7, "ST_CompoundSurface Type and Routines") is supported:

i) The method ST_CompoundSurface(ST_Surface ARRAY) and method ST_CompoundSurface(ST_Surface ARRAY, INTEGER) (Subclause 8.7.2, "ST_CompoundSurface Methods ").

ii) The ST_Surfaces methods (Subclause 8.7.3, "ST_Surfaces Methods").

iii) The ST_ToCompSurface method (Subclause 5.1.55, "Cast").

iv) The CAST ST_Geometry AS ST_CompoundSurface (Subclause 5.1.55, "Cast").

q) If the ST_BRepSolid type (Subclause 9.2.1, "ST_BRepSolid Type Type and Routines") is instantiable:

i) The method ST_BRepSolid (ST_Surface, ST_Surface ARRAY) and the method ST_ BRepSolid(ST_Surface, ST_Surface ARRAY, INTEGER) (Subclause 9.2.2, "ST_BRepSolid Methods").

ii) The ST_InteriorShells methods (Subclause 9.2.4, "ST_InteriorShells Methods").

iii) The ST_ToBRepSolid method (Subclause 5.1.55, "Cast").

iv) The CAST ST_Geometry AS ST_ BRepSolid (Subclause 5.1.55, "Cast").

r) For the ST_GeomCollection type (Subclause 9.1, "ST_GeomCollection Type and Routines"

i) The method ST_GeomCollection(ST_Geometry ARRAY) and the method ST_GeomCollection(ST_Geometry ARRAY, INTEGER) (Subclause 9.1.2, "ST_GeomCollection Methods").

ii) The ST_Geometries methods (Subclause 9.1.3, "ST_Geometries Methods").

iii) The ST_ToGeomColl method (Subclause 5.1.55, "Cast").

iv) The CAST ST_Geometry AS ST_GeomCollection (Subclause 5.1.55, "Cast").

s) If the ST_MultiPoint type (Subclause 9.2, "ST_MultiPoint Type and Routines") is instantiable, then:

i) The method ST_MultiPoint(ST_Point ARRAY) and the method ST_MultiPoint(ST_Point ARRAY, INTEGER) (Subclause 9.2.2, "ST_MultiPoint Methods").

ii) The ST_Geometries methods (Subclause 9.2.3, "ST_Geometries Methods").

iii) The ST_ToMultiPoint method (Subclause 5.1.55, "Cast").

iv) The CAST ST_Geometry AS ST_MultiPoint (Subclause 5.1.55, "Cast").

t) If the ST_MultiCurve type (Subclause 9.3, "ST_MultiCurve Type and Routines") is instantiable, then:
   i) The method ST_MultiCurve(ST_Curve ARRAY) and the method ST_MultiCurve(ST_Curve ARRAY, INTEGER) (Subclause 9.3.2, "ST_MultiCurve Methods").
   ii) The ST_Geometries methods (Subclause 9.3.8, "ST_Geometries Methods").
   iii) The ST_ToMultiCurve method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_MultiLineString (Subclause 5.1.55, "Cast").
u) If the ST_MultiLineString type (Subclause 9.4, "ST_MultiLineString Type and Routines") is instantiable, then:
   i) The method ST_MultiLineString(ST_LineString ARRAY) and the method ST_MultiLineString(ST_LineString ARRAY, INTEGER) (Subclause 9.4.2, "ST_MultiLineString Methods").
   ii) The ST_Geometries methods (Subclause 9.4.3, "ST_Geometries Methods").
   iii) The ST_ToMultiLine method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_MultiLineString (Subclause 5.1.55, "Cast").
v) If the ST_MultiSurface type (Subclause 9.5, "ST_MultiSurface Type and Routines") is instantiable, then:
   i) The method ST_MultiSurface(ST_Surface ARRAY) and the method ST_MultiSurface(ST_Surface ARRAY, INTEGER) (Subclause 9.5.2, "ST_MultiSurface Methods").
   ii) The ST_Geometries methods (Subclause 9.5.11, "ST_Geometries Methods").
   iii) The ST_ToMultiSurface method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_MultiSurface (Subclause 5.1.55, "Cast").
w) If the ST_MultiPolygon type (Subclause 9.6, "ST_MultiPolygon Type and Routines") is instantiable, then:
   i) The method ST_MultiPolygon(ST_Polygon ARRAY) the method ST_MultiPolygon(ST_Polygon ARRAY, INTEGER) (Subclause 9.6.2, "ST_MultiPolygon Methods").
   ii) The ST_Geometries methods (Subclause 9.6.3, "ST_Geometries Methods").
   iii) The ST_ToMultiPolygon method (Subclause 5.1.55, "Cast").
   iv) The CAST ST_Geometry AS ST_MultiPolygon (Subclause 5.1.55, "Cast").
x) If the ST_Vector type (Subclause 16.2, "ST_Vector Type and Routines") is supported:
   i) The ST_Coordinates method (Subclause 16.2.6, "ST_Coordinates Method").
y) For the ST_AffinePlacement type (Subclause 16.3, "ST_AffinePlacement Type and Routines"):
   i) The method ST_AffinePlacement(ST_Point, ST_Vector ARRAY) (Subclause 16.3.2, "ST_AffinePlacement Method").
   ii) The ST_RefDirections methods (Subclause 16.3.4, "ST_RefDirections Methods").

3) Which of the following optional user-defined types and routines are supported:
   a) The method ST_CoordDim() (Subclause 5.1.3, "ST_CoordDim Method").
   b) The method ST_3DIsSimple() (Subclause 5.1.9, "ST_3DIsSimple Method").
   c) The method ST_IsValid() (Subclause 5.1.10, "ST_IsValid Method").
   d) The method ST_Is3D() (Subclause 5.1.11, "ST_Is3D Method").
   e) The method ST_IsMeasured() (Subclause 5.1.12, "ST_IsMeasured Method").

f) The method ST_LocateAlong(DOUBLE PRECISION) (Subclause 5.1.13, "ST_LocateAlong Method").

g) The method ST_3DLocateAlong(DOUBLE PRECISION) (Subclause 5.1.14, "ST_3DLocateAlong Method").

h) The method ST_LocateBetween(DOUBLE PRECISION, DOUBLE PRECISION) (Subclause 5.1.15, "ST_LocateBetween Method").

i) The method ST_3DLocateBetween(DOUBLE PRECISION, DOUBLE PRECISION) (Subclause 5.1.16, "ST_3DLocateBetween Method").

j) The method ST_3DBoundary() (Subclause 5.1.18, "ST_3DBoundary Method").

k) The method ST_MinX() (Subclause 5.1.21, "ST_MinX Method"), the method ST_MaxX() (Subclause 5.1.22, "ST_MaxX Method"), the method ST_MinY() (Subclause 5.1.23, "ST_MinY Method"), and the method ST_MaxY() (Subclause 5.1.24, "ST_MaxY Method").

l) The method ST_MinZ() (Subclause 5.1.25, "ST_MinZ Method") and the method ST_MaxZ() (Subclause 5.1.26, "ST_MaxZ Method").

m) The method ST_MinM() (Subclause 5.1.27, "ST_MinM Method") and the method ST_MaxM() (Subclause 5.1.28, "ST_MaxM Method").

n) The method ST_Buffer(ST_Geometry, CHARACTER VARYING) (Subclause 5.1.30, "ST_Buffer Methods").

o) The method ST_3DIntersection() (Subclause 5.1.32, "ST_3DIntersection Method").

p) The method ST_3DUnion() (Subclause 5.1.34, "ST_3DUnion Method").

q) The method ST_3DDifference() (Subclause 5.1.36, "ST_3DDifference Method").

r) The method ST_3DSymDifference() (Subclause 5.1.38, "ST_3DSymDifference Method").

s) The method ST_Distance(ST_Geometry, CHARACTER VARYING) (Subclause 5.1.41, "ST_Distance Methods").

t) The method ST_3DDistance(ST_Geometry) and the method ST_3DDistance(ST_Geometry, CHARACTER VARYING) (Subclause 5.1.42, "ST_3DDistance Methods").

u) the method ST_3DEquals(ST_Geometry) (Subclause 5.1.44, "ST_3DEquals Method").

v) The method ST_3DDisjoint(ST_Geometry) (Subclause 5.1.47, "ST_3DDisjoint Method").

w) The method ST_3DIntersects(ST_Geometry) (Subclause 5.1.49, "ST_3DIntersects Method").

x) The method ST_GMLToSQL(CHARACTER LARGE OBJECT) (Subclause 5.1.60, "ST_GMLToSQL Method").

y) The method ST_AsGML() (Subclause 5.1.61, "ST_AsGML Method").

z) The function ST_GeomFromGML(CHARACTER LARGE OBJECT) and the function ST_GeomFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 5.1.64, "ST_GeomFromGML Functions").

aa) The method ST_Point(DOUBLE PRECISION, DOUBLE PRECISION), the method ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, INTEGER), the method ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION), the method ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER), the method ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION), the method ST_Point(DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, DOUBLE PRECISION, INTEGER) (Subclause 6.1.2, "ST_Point Methods").

ab) The method ST_X(DOUBLE PRECISION) (Subclause 6.1.3, "ST_X Methods").

ac) The method ST_Y(DOUBLE PRECISION) (Subclause 6.1.4, "ST_Y Methods").

ad) The method ST_Z() and the method ST_Z(DOUBLE PRECISION) (Subclause 6.1.5, "ST_Z Methods").

ae) The method ST_M() and the method ST_M(DOUBLE PRECISION) (Subclause 6.1.6, "ST_M Methods").

af) The method ST_ExplicitPoint() (Subclause 6.1.7, "ST_ExplicitPoint Method").

ag) The function ST_PointFromGML(CHARACTER LARGE OBJECT) and the function ST_PointFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 6.1.10, "ST_PointFromGML Functions").

ah) The method ST_Length(CHARACTER VARYING) (Subclause 7.1.2, "ST_Length Method").

ai) The method ST_3DLength() and the method ST_3DLength(CHARACTER VARYING) (Subclause 7.1.3, "ST_3DLength Methods").

aj) The method ST_3DIsClosed() (Subclause 7.1.7, "ST_3DIsClosed Method").

ak) The method ST_3DIsRing() (Subclause 7.1.9, "ST_3DIsRing Method").

al) The method ST_DistanceToPoint(ST_Point) (Subclause 7.1.11, "ST_DistanceToPoint Methods"), the method ST_DistanceToPoint(ST_Point, CHARACTER VARYING) (Subclause 7.1.11, "ST_DistanceToPoint Methods"), the method ST_PointAtDistance(DOUBLE PRECISION) (Subclause 7.1.13, "ST_PointAtDistance Methods") and the method ST_PointAtDistance(DOUBLE PRECISION, CHARACTER VARYING) (Subclause 7.1.13, "ST_PointAtDistance Methods").

am) The method ST_3DDistanceToPt(ST_Point) (Subclause 7.1.12, "ST_3DDistanceToPt Methods") , the method ST_3DDistanceToPt(ST_Point, CHARACTER VARYING) (Subclause 7.1.12, "ST_3DDistanceToPt Methods"), the method ST_3DPtAtDistance(DOUBLE PRECISION) (Subclause 7.1.14, "ST_3DPtAtDistance Methods") and the method ST_3DPtAtDistance(DOUBLE PRECISION, CHARACTER VARYING) (Subclause 7.1.14, "ST_3DPtAtDistance Methods").

an) The method ST_PerpPoints(ST_Point) (Subclause 7.1.15, "ST_PerpPoints Method"),

ao) The method ST_LineString(CHARACTER LARGE OBJECT), the method ST_LineString(CHARACTER LARGE OBJECT, INTEGER), the method ST_LineString(BINARY LARGE OBJECT), and the method ST_LineString(BINARY LARGE OBJECT, INTEGER) (Subclause 7.2.2, "ST_LineString Methods").

ap) The function ST_LineFromGML(CHARACTER LARGE OBJECT) and the function ST_LineFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 7.2.10, "ST_LineFromGML Functions").

aq) The ST_CircularString type and routines (Subclause 7.3, "ST_CircularString Type and Routines").

ar) The ST_Circle type and routines (Subclause 7.4, "ST_Circle Type and Routines").

as) The ST_GeodesicString type and routines (Subclause 7.5, "ST_GeodesicString Type and Routines").

at) The ST_EllipticalCurve type and routines (Subclause 7.6, "ST_EllipticalCurve Type and Routines").

au) The ST_NURBSCurve type and routines (Subclause 7.7, "ST_NURBSCurve Type and Routines"), ST_NURBPoint type and routines (Subclause 16.4, "ST_NURBSPoint Type and Routines"), and ST_Knot type and routines (Subclause 16.5, "ST_Knot Type and Routines").

av) The ST_Clothoid type and routines (Subclause 7.8, "ST_Clothoid Type and Routines").

aw) The ST_SpiralCurve type and routines (Subclause 7.9, "ST_SpiralCurve Type and Routines").

ax) The ST_CompoundCurve type and routines (Subclause 7.10, "ST_CompoundCurve Type and Routines").

ay) The method ST_Area(CHARACTER VARYING) (Subclause 8.1.2, "ST_Area Methods").

az) The method ST_3DArea() and the method ST_3DArea(CHARACTER VARYING) (Subclause 8.1.3, "ST_3DArea Methods").

ba) The method ST_Perimeter(CHARACTER VARYING) (Subclause 8.1.4, "ST_Perimeter Methods").

bb) The method ST_3DPerimeter() and the method ST_3DPerimeter(CHARACTER VARYING) (Subclause 8.1.5, "ST_3DPerimeter Methods").

bc) The method ST_3DCentroid() (Subclause 8.1.7, "ST_3DCentroid Method").

bd) The method ST_3DPointOnSurf() (Subclause 8.1.7, "ST_3DPointOnSurf Method").

be) The method ST_IsWorld() (Subclause 8.1.10, "ST_IsWorld Method").

bf) The ST_CurvePolygon type and routines (Subclause 8.2, "ST_CurvePolygon Type and Routines").

NOTE The ST_Polygon type inherits the ST_NumInteriorRing method from the ST_CurvePolygon type. If an implementation does not support the ST_CurvePolygon type and routines, then it is mandatory for an implementation to support the ST_NumInteriorRing method on the ST_Polygon type.

bg) The method ST_Polygon(CHARACTER LARGE OBJECT), the method ST_Polygon(CHARACTER LARGE OBJECT, INTEGER), the method ST_Polygon(BINARY LARGE OBJECT), the method ST_Polygon(BINARY LARGE OBJECT, INTEGER), the method ST_Polygon(ST_LineString), and the method ST_Polygon(ST_LineString, INTEGER) (Subclause 8.3.2, "ST_Polygon Methods").

bh) The method ST_ExteriorRing(ST_Curve) (Subclause 8.3.3, "ST_ExteriorRing Methods").

bi) The function ST_PolyFromGML(CHARACTERLARGE OBJECT) and the function ST_PolyFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 8.3.8, "ST_PolyFromGML Functions").

bj) The ST_BdPolyFromText Functions (Subclause 8.3.9, "ST_BdPolyFromText Functions"), the ST_BdPolyFromWKB Functions (Subclause 8.3.10, "ST_BdPolyFromWKB Functions"), the ST_BdMPolyFromText Functions (Subclause 9.6.7, "ST_BdMPolyFromText Functions"), and the ST_BdMPolyFromWKB Functions (Subclause 9.6.8, "ST_BdMPolyFromWKB Functions").

bk) The ST_Triangle type and routines (Subclauses 8.4 "ST_Triangle Type and Routines"), the ST_PolyhdrlSurface type and routines (Subclause 8.5, "ST_PolyhdrlSurface Type and Routines"), the ST_TIN type and routines (Subclause 8.6, "ST_TIN Type and Routines"), and the ST_TINElement types and routines (Subclause 16.1, "ST_TINElement Type and Routines").

bl) The ST_CompoundSurface type and routines (Subclause 8.7, "ST_CompoundSurface Type and Routines).

bm) The ST_Solid type and routines (Subclause 9.1, "ST_Solid Type and Routines) and the ST_BRepSolid type and routines (Subclause 9.2, "ST_BRepSolid Type and Routines).

bn) The method ST_GeomCollection(CHARACTER LARGE OBJECT), the method ST_GeomCollection(CHARACTER LARGE OBJECT, INTEGER), the method ST_GeomCollection(BINARY LARGE OBJECT), the method ST_GeomCollection(BINARY LARGE OBJECT, INTEGER), the method ST_GeomCollection(ST_Geometry), and the method ST_GeomCollection(ST_Geometry, INTEGER) (Subclause 9.1.2, "ST_GeomCollection Methods").

bo) The function ST_GeomCollFromGML(CHARACTER LARGE OBJECT) and the function ST_GeomCollFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 9.1.8, "ST_GeomCollFromGML Functions").

bp) The method ST_MultiPoint(CHARACTER LARGE OBJECT), the method ST_MultiPoint(CHARACTER LARGE OBJECT, INTEGER), the method ST_MultiPoint(BINARY LARGE OBJECT), and the method ST_MultiPoint(BINARY LARGE OBJECT, INTEGER) (Subclause 9.2.2, "ST_MultiPoint Methods").

bq) The function ST_MPointFromGML(CHARACTER LARGE OBJECT) and the function ST_MPointFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 9.2.6, "ST_MPointFromGML Functions").

br) The method ST_MultiCurve(CHARACTER LARGE OBJECT), the method ST_MultiCurve(CHARACTER LARGE OBJECT, INTEGER), the method ST_MultiCurve(BINARY LARGE OBJECT), and the method ST_MultiCurve(BINARY LARGE OBJECT, INTEGER) (Subclause 9.3.2, "ST_MultiCurve Methods").

bs) The method ST_Length(CHARACTER VARYING) (Subclause 9.3.5, "ST_Length Methods").

bt) The method ST_3DIsClosed() (Subclause 9.3.4, "ST_3DIsClosed Method").

bu) The method ST_3DLength() and the method ST_3DLength(CHARACTER VARYING) (Subclause 9.3.6, "ST_3DLength Methods").

bv) The method ST_PerpPoints(ST_Point) (Subclause 9.3.7, "ST_PerpPoints Method"),

bw) The function ST_MCurveFromGML(CHARACTER LARGE OBJECT) and the function ST_MCurveFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 9.3.11, "ST_MCurveFromGML Functions").

bx) The method ST_MultiLineString(CHARACTER LARGE OBJECT), the method ST_MultiLineString(CHARACTER LARGE OBJECT, INTEGER), the method ST_MultiLineString(BINARY LARGE OBJECT), and the method ST_MultiLineString(BINARY LARGE OBJECT, INTEGER) (Subclause 9.4.2, "ST_MultiLineString Methods").

by) The function ST_MLineFromGML(CHARACTER LARGE OBJECT) and the function ST_MLineFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 9.4.6, "ST_MLineFromGML Functions").

bz) The method ST_MultiSurface(CHARACTER LARGE OBJECT), the method ST_MultiSurface(CHARACTER LARGE OBJECT, INTEGER), the method ST_MultiSurface(BINARY LARGE OBJECT), the method ST_MultiSurface(BINARY LARGE OBJECT, INTEGER) (Subclause 9.5.2, "ST_MultiSurface Methods").

ca) The method ST_Area(CHARACTER VARYING) (Subclause 9.5.3, "ST_Area Methods").

cb) The method ST_3DArea() and the method ST_3DArea(CHARACTER VARYING) (Subclause 9.5.4, "ST_3DArea Methods").

cc) The method ST_Perimeter(CHARACTER VARYING) (Subclause 9.5.5, "ST_Perimeter Methods").

cd) The method ST_3DPerimeter() and the method ST_3DPerimeter(CHARACTER VARYING) (Subclause 9.5.6, "ST_3DPerimeter Methods").

ce) The method ST_3DCentroid() (Subclause 9.5.8, "ST_3DCentroid Method").

cf) The method ST_3DPointOnSurf() (Subclause 9.5.10, "ST_3DPointOnSurf Method").

cg) The function ST_MSurfaceFromGML(CHARACTER LARGE OBJECT) and the function ST_MSurfaceFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 9.5.14, "ST_MSurfaceFromGML Functions").

ch) The method ST_MultiPolygon(CHARACTER LARGE OBJECT), the method ST_MultiPolygon(CHARACTER LARGE OBJECT, INTEGER), the method ST_MultiPolygon(BINARY LARGE OBJECT), and the method ST_MultiPolygon(BINARY LARGE OBJECT, INTEGER) (Subclause 9.6.2, "ST_MultiPolygon Methods").

ci) The function ST_MPolyFromGML(CHARACTER LARGE OBJECT) and the function ST_MPolyFromGML(CHARACTER LARGE OBJECT, INTEGER) (Subclause 9.6.6, "ST_MPolyFromGML Functions").

cj) The function ST_ShortestUndPath (Subclause 12.1.1, "ST_ShortestUndPath Function").

ck) The function ST_ShortestDirPath (Subclause 12.1.2, "ST_ShortestDirPath Function").

cl) The ST_Angle type and routines (Subclause 15.1, "ST_Angle Type and Routines").

cm) The ST_Direction type and routines (Subclause 15.2, "ST_Direction Type and Routines").

cn) The ST_Vector type and routines (Subclause 16.2 "ST_Vector Type and Routines").

4) Whether or not the ST_GML transform is supported (Subclause 5.1.66, "SQL Transform Functions").

5) Whether or not the ST_MultiCurve type is instantiable (Subclause 9.3.1, "ST_MultiCurve Type").
6) Whether or not the ST_MultiSurface type is instantiable (Subclause 9.5.1, "ST_MultiSurface Type").
7) Whether or not the ST_CurveToLine method (Subclause 7.1.10, "ST_CurveToLine Method") is supported.
8) Whether or not the optional <linear unit> in <geographic cs> (Subclause 13.1.9, "<spatial reference system>") is supported.
9) The definitions for all elements and actions that this part of ISO/IEC 13249 specified as implementation-defined.
10) Whether or not the Topology-geometry views and routines (Clause 10, "Topology-Geometry") are supported.
11) Whether or not the topology-network views and routines (Clause 11, "Topology-Network") are supported. If the Topology-geometry views and routines (Clause 10, "Topology-Geometry") are not supported, then whether or not the ST_SpatNetFromTGeo Procedure (Subclause 11.3.15, "ST_SpatNetFromTGeo Procedure") is supported.
12) Whether or not the linear referencing types and routines (Clause 14, "Linear Referencing Types") core functionality is supported, including:
    a) Linear Referencing Method ST_LRM values excluding those which support towards referents or lateral, vertical or vector offsets. (Subclause 14.1, "ST_LRM Type and Routines").
    b) Linear Referencing Method ST_LRM attributes and routines excluding ST_PrivateOffsetUnits, ST_PrivatePositiveLateralOffsetDirection and ST_PrivatePositiveVerticalOffsetDirection attributes and related methods. (Subclause 14.1, "ST_LRM Type and Routines").
    c) Linear element ST_LinearElement values. (Subclause 14.2, "ST_LinearElement Type and Routines").
    d) Linearly referenceable feature ST_LRFeature values. (Subclause 14.3, "ST_LRFeature Type and Routines").
    e) Linearly referenceable curve ST_LRCurve values. (Subclause 14.4, "ST_LRCurve Type and Routines").
    f) Linearly referenceable directed edge ST_LRDirectedEdge values. (Subclause 14.5, "ST_LRDirectedEdge Type and Routines").
    g) Position expression ST_PositionExp values excluding those which contain ST_DistanceExp values with towards referents or lateral, vertical or vector offsets. (Subclause 14.6, "ST_PositionExp Type and Routines").
    h) Linear referencing measure ST_LRMeasure values. (Subclause 14.7, "ST_LRMeasure Type and Routines").
    i) Start value ST_StartValue values. (Subclause 14.8, "ST_StartValue Type and Routines").
    j) Distance expression ST_DistanceExp values excluding those which contain towards referents or lateral, vertical or vector offset expressions. (Subclause 14.9, "ST_DistanceExp Type and Routines").
    k) Referent ST_Referent values. (Subclause 14.10, "ST_Referent Type and Routines").
13) Whether or not the linear referencing types and routines (Clause 14, "Linear Referencing Types") towards referent extension is supported (in addition to the core linear referencing types and routines), including:
    a) Linear Referencing Method ST_LRM values which support towards referents. (Subclause 14.1, "ST_LRM Type and Routines").
    b) Position expression ST_PositionExp values which contain ST_DistanceExp values with towards referents. (Subclause 14.6, "ST_PositionExp Type and Routines").
    c) Distance expression ST_DistanceExp ST_PrivateTowardsReferentFeatureID and ST_PrivateTowardsReferentID attributes and related methods. (14.9, "ST_DistanceExp Type and Routines").

14) Whether or not the linear referencing types and routines (Clause 14, "Linear Referencing Types") lateral and vertical offset extension is supported (in addition to the core linear referencing types and routines), including:
    a) Linear Referencing Method ST_LRM values which support lateral and vertical offsets. (Subclause 14.1, "ST_LRM Type and Routines").
    b) Linear Referencing Method ST_LRM ST_PrivateOffsetUnits, ST_PrivatePositiveLateralOffsetDirection and ST_PrivatePositiveVerticalOffsetDirection attributes and related methods. (Subclause 14.1, "ST_LRM Type and Routines").
    c) Position expression ST_PositionExp values which contain ST_DistanceExp values with lateral or vertical offsets. (Subclause 14.6, "ST_PositionExp Type and Routines").
    d) Distance expression ST_DistanceExp ST_PrivateLateralOffsetExpression and ST_PrivateVerticalOffsetExpression attributes and related methods. (Subclause 14.9, "ST_DistanceExp Type and Routines").
    e) Lateral offset expression ST_LatOffsetExp type and routines. (Subclause 14.11, "ST_LatOffsetExp Type and Routines").
    f) Vertical offset expression ST_VerOffsetExp type and routines. (Subclause 14.12, "ST_VerOffsetExp Type and Routines").
15) Whether or not the linear referencing types and routines (Clause 14, "Linear Referencing Types") vector offset extension is supported (in addition to the core linear referencing types and routines and the lateral and vertical offset extension), including:
    a) Position expression ST_PositionExp values which contain ST_DistanceExp values with vector offsets. (Subclause 14.6, "ST_PositionExp Type and Routines").
    b) Distance expression ST_DistanceExp ST_PrivateVectorOffsetExpression attributes and related methods. (Subclause 14.9, "ST_DistanceExp Type and Routines").
    c) Vector offset expression ST_VectorOffsetExp type and routines. (Subclause 14.13, "ST_VectorOffsetExp Type and Routines").

Annex A

(informative)

Implementation-defined elements

This Annex references those features that are identified in the body of this part of ISO/IEC 13249 as implementation-defined.

The term implementation-defined is used to identify characteristics that may differ between implementations, but that shall be defined for each particular implementation.

1) Subclause 3.2.2, " Notations provided in Part 3"

   a) Paragraph 3)

      The real number mathematical constant that represents the circumference of a circle with unit diameter is notated as “π”. This number is transcendental and cannot be represented exactly in any algebraic form, so the precision is implementation-defined.

2) Subclause 4.2.2.3, "Spatial Methods using ST_Geometry"

   a) List item 1)

      An implementation-defined tolerance may be provided such that two points are considered equal if the distance between the points is less that the tolerance.

3) Subclause 4.2.19, "ST_TIN"

   a) Paragraph 1)

      The ST_TIN type is a subtype of ST_PolyhdrlSurface composed only of triangles (ST_Triangle) that uses the Delaunay algorithm [3], or a similar implementation-defined algorithm, complemented with consideration for breaklines, soft breaks, control contours, break voids, drape voids, voids, holes, stop lines and maximum length of triangle sides. The ST_TIN type is instantiable.

4) Subclause 4.12, "The Spatial Information Schema"

   a) List item 4)

      a view ST_SIZINGS, which lists implementation-defined meta-variables and their values.

5) Subclause 5.1.4, "ST_GeometryType Method"

   a) Description 2) h)

      Otherwise, the method *ST_GeometryType()* returns an implementation-defined CHARACTER VARYING value for a user-defined type not defined in this part of ISO/IEC 13249.

6) Subclause 5.1.6, "ST_Transform Method"

   a) Description 3) d)

      Otherwise, return an *ST_Geometry* value as the result of an implementation-defined transform of SELF from the spatial reference system of SELF to the spatial reference system specified by *ansrid*. The value returned has the spatial reference system identifier equal to *ansrid*.

7) Subclause 5.1.15, "ST_LocateBetween Method"

   a) Description 2) e)

      If *SELF.ST_Dimension*() is equal to 1, then use the implementation-defined interpolation algorithm to estimate values between *start_measure* and *end_measure* inclusively.

   b) Description 2) e) iii) 1) A)

If the result also contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then it is implementation-defined whether or not the following completion condition is raised: *SQL/MM Spatial warning – disconnected points not included in result*

c) Description 2) f)

If *SELF.ST_Dimension*() is equal to 2, then the operation is implementation-defined.

8) Subclause 5.1.16, "ST_3DLocateBetween Method"

a) Description 2) e)

If *SELF.ST_Dimension*() is equal to 1, then use the implementation-defined interpolation algorithm to estimate values between *start_measure* and *end_measure* inclusively.

b) Description 2) e) iii) 1) A)

If the result also contains disconnected points with m coordinate values between *start_measure* and *end_measure* inclusively, then it is implementation-defined whether or not the following completion condition is raised: *SQL/MM Spatial warning – disconnected points not included in result*.

c) Description 2) f)

If *SELF.ST_Dimension*() is equal to 2, then the operation is implementation-defined.

9) Subclause 5.1.19, "ST_Envelope Method"

a) Description 2) c)

Let *ETOL* be an implementation-defined envelope tolerance. *ETOL* shall be greater than zero.

10) Subclause 5.1.30, "ST_Buffer Methods"

a) Description 2) a)

The parameter *adistance* is measured in an implementation-defined linear unit of measure in the spatial reference system of SELF.

11) Subclause 5.1.41, "ST_Distance Methods"

a) Description 2) a) iv)

The distance between the two points is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_Distance(ST_Geometry)* is in an implementation-defined unit of measure.

c) Description 4) d) iv)

The distance between the two points is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.

12) Subclause 5.1.42, "ST_3DDistance Methods"

a) Description 2) a) iv)

The distance between the two points is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DDistance(ST_Geometry)* is in an implementation-defined unit of measure.

c) Description 4) d) iv)

The distance between the two points is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are not considered in the calculation.

13) Subclause 5.1.56, "ST_WKTToSQL Method"

   a) Description 2)

      If *awkt* is not producible in the BNF for <well-known text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

14) Subclause 5.1.58, "ST_WKBToSQL Method"

   a) Description 2)

      If *awkb* is not producible in the BNF for <well-known binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

15) Subclause 5.1.60, "ST_GMLToSQL Method"

   a) Description 2)

      If *agml* does not contain an XML element as defined in Table 14 — Mapping between ST_Geometry values and GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

16) Subclause 5.1.62, "ST_GeomFromText Functions"

   a) Description 4) a)

      If *awkt* is not producible in the BNF for <well-known text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

17) Subclause 5.1.63, "ST_GeomFromWKB Functions"

   a) Description 4) a)

      The parameter *awkb* is the well-known binary representation of an *ST_Geometry* value. If *awkb* is not producible in the BNF for <well-known binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

18) Subclause 5.1.64, "ST_GeomFromGML Functions"

   a) Description 4) a)

      If the parameter *agml* does not contain an XML element as defined in Table 14 — Mapping between ST_Geometry values and GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

19) Subclause 6.1.8, "ST_PointFromText Functions"

   a) Description 4) a)

      If *awkt* is not producible in the BNF for <point text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

20) Subclause 6.1.9, "ST_PointFromWKB Functions"

   a) Description 4) a)

      If *awkb* is not producible in the BNF for <point binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

21) Subclause 6.1.10, "ST_PointFromGML Functions"

   a) Description 4) a)

If the parameter *agml* does not contain a Point XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

22) Subclause 7.1.2, "ST_Length Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined length of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined length of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

23) Subclause 7.1.3, "ST_3DLength Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined length of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DLength()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined length of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

24) Subclause 7.1.10, "ST_CurveToLine Method"

a) Description 2) b)

Otherwise, return the implementation-defined *ST_LineString* value approximation of the *ST_Curve* value.

b) Description 4)

If *SELF.ST_IsMeasured()* is equal to 1 (one), then m coordinate values are calculated for the *ST_LineString.ST_PrivatePoints ST_Point* values by linear interpolation based on curve length using an implementation-defined interpolation algorithm. The resultant m coordinate values are included in the resultant geometry.

25) Subclause 7.1.11, "ST_DistanceToPoint Methods"

a) Description 2) a) iv)

Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculations and return value.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_DistanceToPoint(ST_Point)* is in an implementation-defined unit of measure.

c) Description 4) d) iv)

Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculations and return value.

26) Subclause 7.1.12, "ST_3DDistanceToPt Methods"

a) Description 2) a) iv)

Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are considered in the calculations and return value.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DDistanceToPt(ST_Point)* is in an implementation-defined unit of measure.

c) Description 4) d) iv)

Otherwise, return the distance along the curve SELF from the start point to *apoint*, calculated in the spatial reference system of SELF. The distance along the curve is calculated using an implementation-defined algorithm such that z (but not m) coordinate values are considered in the calculations and return value.

27) Subclause 7.1.13, "ST_PointAtDistance Methods"

a) Description 2) a)

The parameter *adistance* is measured in an implementation-defined linear unit of measure in the spatial reference system of SELF.

b) Description 2) b) iii)

Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are not considered in the calculations and an interpolated m (but not z) coordinate is included in the return value.

c) Description 4) d) iii)

Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are not considered in the calculations and an interpolated m (but not z) coordinate is included in the return value.

28) Subclause 7.1.14, "ST_3DPtAtDistance Methods"

a) Description 2) a)

The parameter *adistance* is measured in an implementation-defined linear unit of measure in the spatial reference system of SELF.

b) Description 2) b) iii)

Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are considered in the calculations and a z and interpolated m coordinate is included in the return value.

c) Description 4) d) iii)

Otherwise, return the point that is *adistance* along the curve SELF from the start point, calculated in the spatial reference system of SELF. The point that is *adistance* along the curve is determined by using an implementation-defined algorithm such that z coordinate values are considered in the calculations and a z and interpolated m coordinate is included in the return value.

29) Subclause 7.1.15, "ST_PerpPoints Method"

a) Description 2) e)

Otherwise, return a geometry value representing the perpendicular projection of *apoint* on SELF, calculated in the spatial reference system of SELF, using an implementation-defined algorithm such that z coordinate values are not considered in the calculation and interpolated m (but not z) coordinates are included in the return values.

NOTE The result of the projection algorithm may produce the following
- an ST_Point value when it produces a single point result
- an ST_MultiPoint value when it produces a finite number of points
- an ST_Curve value when it produces a connected set of points
- an ST_MultiCurve value when it produces a number of connected set of points
- an ST_GeomCollection when it produces a mixture of point values and curve values.

30) Subclause 7.2.8, "ST_LineFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <linestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

31) Subclause 7.2.9, "ST_LineFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <linestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

32) Subclause 7.2.10, "ST_LineFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a LineString or LineStringSegment XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

33) Subclause 7.3.12, "ST_Radius Method"

a) Description 2) a) iii)

Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_Radius()* is in an implementation-defined unit of measure.

c) Description 4) d) iii)

Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

34) Subclause 7.3.17, "ST_CircularFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <circularstring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

35) Subclause 7.3.18, "ST_CircularFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <circularstring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

36) Subclause 7.3.20, "ST_CircularFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain an Arc, ArcString, ArcByBulge, ArcStringByBulge or ArcByCenterPoint XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

37) Subclause 7.4.5, "ST_Radius Method"

a) Description 2) a) ii)

Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_Radius()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined radius of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

38) Subclause 7.4.10, "ST_CircleFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <circle text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

39) Subclause 7.4.11, "ST_CircleFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <circle binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

40) Subclause 7.4.12, "ST_CircleFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Circle or CircleByCenterPoint XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

41) Subclause 7.5.8, "ST_GeodesicFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <geodesic text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

42) Subclause 7.5.9, "ST_GeodesicFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <geodesic binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

43) Subclause 7.5.10, "ST_GeodesicFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Geodesic or GeodesicString XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

44) Subclause 7.6.2, "ST_EllipticalCurve Methods"

a) Description 16) b)

Otherwise, the values *auaxislength* and *avaxislength* are in an implementation-defined unit of measure.

45) Subclause 7.6.4, "ST_UAxisLength Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_UAxisLength()* is in an implementation-defined unit of measure.

b) Description 7) b) ii)

Otherwise, the value returned by *ST_UAxisLength()* is in an implementation-defined unit of measure.

46) Subclause 7.6.5, "ST_VAxisLength Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_VAxisLength()* is in an implementation-defined unit of measure.

b) Description 7) b) ii)

Otherwise, the value returned by *ST_VAxisLength()* is in an implementation-defined unit of measure.

47) Subclause 7.6.12, "ST_EllipticFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <elliptical text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

48) Subclause 7.6.13, "ST_EllipticFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <elliptical binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

49) Subclause 7.6.14, "ST_EllipticFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain an EllipticalCurve or Ellipse XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

50) Subclause 7.7.10, "ST_NURBSFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <nurbs text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

51) Subclause 7.7.11, "ST_NURBSFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <nurbs binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

52) Subclause 7.7.12, "ST_NURBSFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a BSpline XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

53) Subclause 7.8.2, "ST_Clothoid Methods"

a) Description 15) b)

Otherwise, the values *astartdistance* and *anenddistance* are in an implementation-defined unit of measure.

54) Subclause 7.8.5, "ST_StartDistance Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_StartDistance()* is in an implementation-defined unit of measure.

b) Description 7) b) ii)

Otherwise, the value returned by *ST_StartDistance()* is in an implementation-defined unit of measure.

55) Subclause 7.8.6, "ST_EndDistance Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_EndDistance()* is in an implementation-defined unit of measure.

b) Description 7) b) ii)

Otherwise, the value returned by *ST_EndDistance()* is in an implementation-defined unit of measure.

56) Subclause 7.8.11, "ST_ClothoidFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <clothoid text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

57) Subclause 7.8.12, "ST_ClothoidFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <clothoid binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

58) Subclause 7.8.13, "ST_ClothoidFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain an Clothoid XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

59) Subclause 7.9.1, "ST_SpiralCurve Type"

a) Description 19

The curvature function for the specific spiral type shall be implementation-defined.

60) Subclause 7.9.2, "ST_SpiralCurve Methods"

a) Description 16) b)

Otherwise, the value *alength* is in an implementation-defined unit of measure.

61) Subclause 7.9.4, "ST_Length Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.

b) Description 7) b) ii)

Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.

62) Subclause 7.9.12, "ST_SpiralFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <spiral text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

63) Subclause 7.9.13, "ST_SpiralFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <spiral binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

64) Subclause 7.9.14, "ST_SpiralFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a SpiralCurve XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

65) Subclause 7.10.8, "ST_CompoundFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <compoundcurve text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

66) Subclause 7.10.9, "ST_CompoundFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <compoundcurve binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

67) Subclause 7.10.10, "ST_CompoundFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a CompositeCurve XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

68) Subclause 8.1.2, "ST_Area Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined area of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_Area()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined area of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

69) Subclause 8.1.3, "ST_3DArea Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined area of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DArea()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined area of SELF, such that z (but not m) coordinate values are considered in the calculation, as measured in its spatial reference system.

70) Subclause 8.1.4, "ST_Perimeter Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined length of the boundary of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_Perimeter()* is in an implementation-defined unit of measure.

a) Description 4) d) ii)

Otherwise, return the implementation-defined length of the boundary of SELF, such that z and m coordinate values are not considered in the calculation, as measured in its spatial reference system.

71) Subclause 8.1.5, "ST_3DPerimeter Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined length of the boundary of SELF, such that z (but not m) coordinate values are not considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DPerimeter()* is in an implementation-defined unit of measure.

a) Description 4) d) ii)

Otherwise, return the implementation-defined length of the boundary of SELF, such that z (but not m) coordinate values are not considered in the calculation, as measured in its spatial reference system.

72) Subclause 8.2.7, "ST_CurvePolyToPoly Method"

a) Description 2) b)

Otherwise, return the implementation-defined *ST_Polygon* value approximation of the *ST_CurvePolygon* value.

b) Description 4)

If *SELF.ST_IsMeasured()* is equal to 1 (one), then m coordinate values are calculated for the *ST_Curve.ST_PrivatePoints ST_Point* values by linear interpolation based on curve length using an implementation-defined interpolation algorithm. The resultant m coordinate values are included in the resultant geometry.

73) Subclause 8.2.8, "ST_CPolyFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <curvepolygon text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

74) Subclause 8.2.9, "ST_CPolyFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <curvepolygon binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

75) Subclause 8.2.10, "ST_CPolyFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Polygon or PolygonPatch XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

76) Subclause 8.3.6, "ST_PolyFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <polygon text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

77) Subclause 8.3.7, "ST_PolyFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <polygon binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

78) Subclause 8.3.8, "ST_PolyFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Polygon or PolygonPatch XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

b) Description 4) b) i)

if any of the Polygon or PolygonPatch XML element Rings are not linear, convert them into their implementation-defined LinearRing approximations.

79) Subclause 8.3.9, "ST_BdPolyFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multilinestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

80) Subclause 8.3.10, "ST_BdPolyFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multilinestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

81) Subclause 8.4.4, "ST_3DSlope Method"

a) Description 2) a) iii)

Otherwise, return the implementation-defined slope of SELF, such that z coordinate values are considered in the calculation.

82) Subclause 8.4.8, "ST_TriFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <triangle text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

83) Subclause 8.4.9, "ST_TriFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <triangle binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

84) Subclause 8.4.10, "ST_TriFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Triangle XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

85) Subclause 8.5.6, "ST_PhSFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <polyhedralsurface text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

86) Subclause 8.5.7, "ST_PhSFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <polyhedralsurface binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

87) Subclause 8.5.8, "ST_PhSFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a PolyhedralSurface or PolygonPatch XML element in the GML representation , then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

88) Subclause 8.6.1, "ST_TIN Type"

a) Definitional Rules 5)

DT1 is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.

b) Description 6)

It is implementation-defined whether the restriction imposed by the *ST_PrivateMaxSideLength* attribute applies to all of the *ST_Triangle* values in the *ST_PrivatePatches* attribute or just those which lie along the boundary of the TIN surface.

89) Subclause 8.6.2, "ST_TIN Methods"

a) Description 14) b)

Using the method *ST_Patches(ST_Triangle ARRAY)*, the *ST_PrivatePatches* attribute set to *triangles*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(triangles)* where *triangles* is an *ST_Triangle* ARRAY obtained by applying the implementation-defined triangulation algorithm to the *ST_TINElement* ARRAY *elements* and constrained or modified by *maxsidelength*.

b) Description 16) b)

Using the method *ST_Patches(ST_Triangle ARRAY)*, the *ST_PrivatePatches* attribute set to *triangles*, the *ST_PrivateDimension* attribute set to 2, and the *ST_PrivateCoordinateDimension* attribute set to *ST_GetCoordDim(triangles)* where *triangles* is an *ST_Triangle* ARRAY obtained by applying the implementation-defined triangulation algorithm to the *ST_TINElement* ARRAY *elements* and constrained or modified by *maxsidelength*.

90) Subclause 8.6.3, "ST_TINElements Methods"

a) Description 4) c) i) 3)

update *triangles* by applying the implementation-defined triangulation algorithm to the *ST_TINElements* ARRAY *elements* and constrained or modified by *maxsidelength.*

b) Description 5)

It is implementation-defined which of the predefined TIN element types are supported, which additional TIN element types are supported, what type of *ST_Geometry* each requires, what behavior is to be expected during triangulation and what exceptions might be raised.

91) Subclause 8.6.4, "ST_MaxSideLength Methods"

a) Description 4) c) i) 3)

update *triangles* by applying the implementation-defined triangulation algorithm to the *ST_TINElements* ARRAY *elements* and constrained or modified by *maxsidelength*

92) Subclause 8.6.5 "ST_TINTable Methods"

a) Definitional Rules 3)

DT1 is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.

b) Description 5) d)

If *CARDINALITY(triangles)* = 0, create *ST_Triangles* by applying the implementation-defined triangulation algorithm to the *ST_Point* ARRAY *points* and constrained or modified by *elements* and *maxsidelength*.

93) Subclause 8.6.8, "ST_TINFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <tin text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*

94) Subclause 8.6.9, "ST_TINFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <tin binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

95) Subclause 8.6.10, "ST_TINFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a TIN XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

96) Subclause 8.7.6, "ST_CompSurfFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <compoundsurface text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*

97) Subclause 8.7.7, "ST_CompSurfFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <compoundsurface binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

98) Subclause 8.7.8, "ST_CompSurfFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a CompositeSurface XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

99) Subclause 9.1.2, "ST_3DSurfaceArea Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined surface area of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DSurfaceArea()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined area of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.

100) Subclause 9.1.3, "ST_3DVolume Methods"

a) Description 2) a) ii)

Otherwise, return the implementation-defined volume of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.

b) Description 2) b) ii)

Otherwise, the value returned by *ST_3DVolume()* is in an implementation-defined unit of measure.

c) Description 4) d) ii)

Otherwise, return the implementation-defined volume of SELF, such that z coordinate values are considered in the calculation, as measured in its spatial reference system.

101) Subclause 9.2.7, "ST_BRepFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <brepsolid text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

102) Subclause 9.2.8, "ST_BRepFromWKB Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <brepsolid binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

103) Subclause 9.2.7, "ST_BRepFromText Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Solid XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

104) Subclause 9.1.6, "ST_GeomCollFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <collection text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

105) Subclause 9.1.7, "ST_GeomCollFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <collection binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

106) Subclause 9.1.8, "ST_GeomCollFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a MultiGeometry XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

107) Subclause 9.2.4, "ST_MPointFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multipoint text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

108) Subclause 9.2.5, "ST_MPointFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multipoint binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

109) Subclause 9.2.6, "ST_MPointFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a MultiPoint XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

110) Subclause 9.3.5, "ST_Length Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_Length()* is in an implementation-defined unit of measure.

111) Subclause 9.3.6, "ST_3DLength Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_3DLength()* is in an implementation-defined unit of measure.

112) Subclause 9.3.7, "ST_PerpPoints Method"

a) Description 2) a) v)

Otherwise, return a geometry value representing the perpendicular projection of *apoint* on SELF, calculated in the spatial reference system of SELF, using an implementation-defined algorithm such that z and m coordinate values are not considered in the calculation or in the return values.

NOTE The result of the projection algorithm may produce the following
- an ST_Point value when it produces a single point result
- an ST_MultiPoint value when it produces a finite number of points
- an ST_Curve value when it produces a connected set of points
- an ST_MultiCurve value when it produces a number of connected set of points
- an ST_GeomCollection when it produces a mixture of point values and curve values.

113) Subclause 9.3.9, "ST_MCurveFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multicurve text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

114) Subclause 9.3.10, "ST_MCurveFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multicurve binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

115) Subclause 9.3.11, "ST_MCurveFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a MultiCurve XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

116) Subclause 9.4.4, "ST_MLineFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multilinestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

117) Subclause 9.4.5, "ST_MLineFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multilinestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

118) Subclause 9.4.6, "ST_MLineFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a MultiLineString XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

119) Subclause 9.5.3, "ST_Area Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_Area()* is in an implementation-defined unit of measure.

120) Subclause 9.5.4, "ST_3DArea Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_3DArea()* is in an implementation-defined unit of measure.

121) Subclause 9.5.5, "ST_Perimeter Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_Perimeter()* is in an implementation-defined unit of measure.

122) Subclause 9.5.6, "ST_3DPerimeter Methods"

a) Description 2) b) ii)

Otherwise, the value returned by *ST_3DPerimeter()* is in an implementation-defined unit of measure.

123) Subclause 9.5.12, "ST_MSurfaceFromTxt Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multisurface text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

124) Subclause 9.5.13, "ST_MSurfaceFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multisurface binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

125) Subclause 9.5.14, "ST_MSurfaceFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a MultiSurface XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

126) Subclause 9.6.4, "ST_MPolyFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multipolygon text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

127) Subclause 9.6.5, "ST_MPolyFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multipolygon binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

128) Subclause 9.6.6, "ST_MPolyFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a MultiPolygon XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

129) Subclause 9.6.7, "ST_BdMPolyFromText Functions"

a) Description 4) a)

If *awkt* is not producible in the BNF for <multilinestring text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

130) Subclause 9.6.8, "ST_BdMPolyFromWKB Functions"

a) Description 4) a)

If *awkb* is not producible in the BNF for <multilinestring binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

131) Subclause 12.1.1, "ST_ShortestUndPath Function"

a) Decription 2)

DT1 is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.

b) Description 3)

The value of *total_length* is measured in an implementation-defined linear unit of measure of the spatial reference system of SELF.

132) Subclause 12.1.2, "ST_ShortestDirPath Function"

a) Decription 2)

DT1 is data type of variable-length character string with character set SQL_IDENTIFIER and implementation-defined maximum length.

b) Description 3)

The value of *total_weight* is measured in an implementation-defined linear unit of measure of the spatial reference system of SELF.

133) Subclause 13.1.2, "ST_SpatialRefSys Methods"

a) Description 2)

If *awkt* is not producible in the BNF for <spatial reference system>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

134) Subclause 13.1.4, "ST_WKTSRSToSQL Method"

a) Description 2)

If *awkt* is not producible in the BNF for <spatial reference system>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

135) Subclause 13.1.5, "ST_SRID Method"

a) Description 2)

A spatial reference system identifier that is equal to 0 (zero) is implementation-defined.

136) Subclause 13.1.6, "ST_Equals Method"

a) Description 3)

The method *ST_Equals(ST_SpatialRefSys)* is implementation-defined.

137) Subclause 13.1.9, "<spatial reference system>"

a) Description 1) b)

The coordinate reference system support for *ST_Geometry* values with m coordinate values is implementation-defined.

b) Description 1) d)

<projection name> is an implementation-defined name of a parameter.

c) Description 1) i)

<parameter name> is an implementation-defined name of a parameter.

d) Description 1) j)

<datum name> is an implementation-defined name of a datum.

e) Description 1) k)

<spheroid>s are implementation-defined.

f) Description 1) l)

<prime meridian>s are implementation-defined.

g) Description 1) m)

<angular unit>s and <linear unit>s are implementation-defined.

138) Subclause 14.1.11, "ST_LRMFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <lrm text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

139) Subclause 14.1.12, "ST_LRMFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a LinearReferencingMethod XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

140) Subclause 14.2.1, "ST_LinearElement Type"

a) Description 12)

Position expression translation is dependent upon the mappings defined between linear elements and LRMs. It is implementation-defined how these mappings are defined and how the system chooses which one(s) to use for a given position expression translation.

a) Description 13)

For the method *ST_TranslateToInst*, if the source and target linear elements are collinear or intersecting at the location specified by the source position expression, then the returned linearly referenced location shall be spatially equal to the linearly referended location specified by the source *ST_PositionExp*. Otherwise, it shall be implementation-defined whether the translation should follow a normal from the source or the target in order to insure that the commutative relationship holds.

141) Subclause 14.2.9, "ST_LEFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <linear element text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

142) Subclause 14.2.10, "ST_LEFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

143) Subclause 14.3.5, "ST_LRFeatFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <lr feature text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

144) Subclause 14.3.6, "ST_LRFeatFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a feature type of LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

145) Subclause 14.4.6, "ST_LRCurveFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <lr curve text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

146) Subclause 14.4.7, "ST_LRCurveFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a curve type of LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

147) Subclause 14.5.6, "ST_LREdgeFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <lr directed edge text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

148) Subclause 14.5.7, "ST_LREdgeFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain an edge type of LinearElement XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

149) Subclause 14.6.9, "ST_PosExpFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <position expression text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

150) Subclause 14.6.10, "ST_PosExpFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a PositionExpression XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

151) Subclause 14.9.11, "ST_DisExpFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <distance expression text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

152) Subclause 14.9.12, "ST_DisExpFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a DistanceExpression XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

153) Subclause 15.1.8, "ST_String Methods"

a) Description 2)

The choice of rounding or truncating is implementation-defined.

b) Description 4)

The choice of rounding or truncating is implementation-defined.

c) Description 5) b)

The maximum measure value for *numdecdigits* is implementation-defined.

154) Subclause 15.1.15, "ST_GMLToSQL Method"

a) Description 2)

The parameter *agml* is the GML Angle representation of an *ST_Angle* value. If *agml* does not contain an Angle XML element, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

155) Subclause 15.1.17, "ST_AngleFromText Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <angle text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

156) Subclause 15.1.18, "ST_AngleFromGML Function"

a) Description 2) a)

If the parameter *agml* does not contain an Angle XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

157) Subclause 15.2.6, "ST_GMLToSQL Method"

a) Description 2)

The parameter *agml* is the GML Direction representation of an *ST_Direction* value. If *agml* does not contain a Direction XML element in the GML representation that can be transformed into an ST_Direction value, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

158) Subclause 15.2.8, "ST_RadianBearing Method"

a) Description 5) c)

The choice of rounding or truncating is implementation-defined.

b) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

159) Subclause 15.2.9, "ST_DegreesBearing Method"

a) Description 5) c)

The choice of rounding or truncating is implementation-defined.

b) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

160) Subclause 15.2.10, "ST_DMSBearing Method"

a) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

161) Subclause 15.2.11, "ST_RadianNAzimuth Method"

a) Description 5) c)

The choice of rounding or truncating is implementation-defined.

b) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

162) Subclause 15.2.12, "ST_DegreesNAzimuth Method"

a) Description 5) c)

The choice of rounding or truncating is implementation-defined.

b) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

163) Subclause 15.2.13, "ST_DMSNAzimuth Method"

a) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

164) Subclause 15.2.14, "ST_RadianSAzimuth Method"

a) Description 5) c)

The choice of rounding or truncating is implementation-defined.

b) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

165) Subclause 15.2.15, "ST_DegreesSAzimuth Method"

a) Description 5)c)

The choice of rounding or truncating is implementation-defined.

b) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

166) Subclause 15.2.16, "ST_DMSSAzimuth Method"

a) Description 6)

The maximum measure value of *numdecdigits* is implementation-defined.

167) Subclause 15.2.19, "ST_DirectionFrmTxt Function"

a) Description 2) a)

If *awkt* is not producible in the BNF for <direction text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

168) Subclause 15.2.20, "ST_DirectionFrmGML Function"

a) Description 2) a)

If the parameter *agml* does not contain a Direction XML element in the GML representation that can be transformed into an ST_Direction value, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

169) Subclause 16.1.1, "ST_TINElement Type"

a) Description 5)

It is implementation-defined which of the predefined TIN element types are supported.

b) Description 6)

It is implementation-defined which additional TIN element types are supported, what type of *ST_Geometry* each requires, and what behavior is to be expected during triangulation.

c) Description 9) l)

otherwise, *SELF.ST_PrivateElementGeometry.ST_GeometryType* and *SELF.ST_PrivateElementGeometry.ST_Is3D* are implementation-defined.

d) Description 12)

The TIN element having a 'boundary' element type can be used to define the boundary in the resulting *ST_TIN* value. When supplied to the *ST_TINElements* mutator method of the *ST_TIN*, the TIN surface value is clipped to the *ST_TINElement ST_Polygon* value. There can be at most only one such element for each *ST_TIN* value. It is implementation-defined whether interior boundaries are supported.

e) Description 25)

It is implementation-defined whether the *ST_Triangle* values in the *ST_PrivatePatches* attribute whose boundaries are crossed by a stop line are removed from the *ST_PrivatePatches* attribute collection of *ST_Triangle* values. If they remain in the collection, it is implementation-defined whether they are enclosed within a 'drape void' type of *ST_TinElement*.

170) Subclause 16.2.11, "ST_WKTToSQL Method"

a) Description 2)

The parameter *awkt* is the well-known text representation of an *ST_Vector* value. If *awkt* is not producible in the BNF for <well-known text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

171) Subclause 16.2.13, "ST_WKBToSQL Method"

a) Description 2)

The parameter *awkb* is the well-known binary representation of an *ST_Vector* value. If *awkb* is not producible in the BNF for <well-known binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

172) Subclause 16.2.15, "ST_GMLToSQL Method"

a) Description 2)

The parameter *agml* is the GML representation of an *ST_Vector* value. If *agml* does not contain a Vector XML element , then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

173) Subclause 16.2.17, "ST_VectorFromText Functions"

a) Description 4) a)

The parameter *awkt* is the well-known text representation of an *ST_Vector* value. If *awkt* is not producible in the BNF for <vector text representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known text representation*.

174) Subclause 16.2.18, "ST_VectorFromWKB Functions"

a) Description 4) a)

The parameter *awkb* is the well-known binary representation of an *ST_Vector* value. If *awkb* is not producible in the BNF for <vector binary representation>, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid well-known binary representation*.

175) Subclause 16.2.19, "ST_VectorFromGML Functions"

a) Description 4) a)

If the parameter *agml* does not contain a Vector XML element in the GML representation, then it is implementation-defined whether or not the following exception condition is raised: *SQL/MM Spatial Exception – invalid GML representation*.

176) Subclause 18.1,"Introduction"

a) Paragraph 1)

How these views are updated is implementation-defined.

A.1 Implementation-defined Meta-variables

1) *ST_ApproximatePi* is the implementation-defined meta-variable representing π.

2) *ST_MaxAngleAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Angle* value.

3) *ST_MaxAngleAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Angle* value.

4) *ST_MaxAngleString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Angle* value.

5) *ST_MaxArrayElements* is the implementation-defined maximum cardinality of an array for the number of geometric paths.

6) *ST_MaxConstraintArrayElements* is the implementation-defined maximum cardinality of an array of CHARACTER VARYING constraint values.

7) *ST_MaxConstraintLength* is the implementation-defined maximum length of the CHARACTER VARYING used for an LRM constraint.

8) *ST_MaxDescriptionLength* is the implementation-defined maximum length used for the character representation of a description.

9) *ST_MaxDirectionAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Direction* value.

10) *ST_MaxDirectionAsText* is the implementation-defined maximum length of the CHARACTER VARYING used for the well-known text representation of an *ST_Direction* value.

11) *ST_MaxDirectionString* is the implementation-defined maximum length of the CHARACTER VARYING used for the character string representation of an *ST_Direction* value.

12) *ST_MaxDoublePrecisionArrayElements* is the implementation-defined maximum cardinality of an array of DOUBLE PRECISION elements.

13) *ST_MaxFeatureIDLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the identification of a linearly referenceable feature.

14) *ST_MaxGeometryArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Geometry* values.

15) *ST_MaxGeometryAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Geometry* value.

16) *ST_MaxGeometryAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Geometry* value.

17) *ST_MaxGeometryAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation (and sometimes the GML representation) of an *ST_Geometry* value.

18) *ST_MaxIntegerArrayElements* is the implementation-defined maximum cardinality of an array of INTEGER elements.

19) *ST_MaxKnotArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Knot* values.

20) *ST_MaxLEMeasureArrayElements* is the implementation-defined maximum cardinality of an array of *ST_LEMeasure* values.

21) *ST_MaxLRAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of a linear referencing type value.

22) *ST_MaxLRAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representations of a linear referencing type value.

23) *ST_MaxLRMNameLength* is the implementation-defined maximum length of the CHARACTER VARYING used for the name of a linear referencing method.

24) *ST_MaxNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING network name.

25) *ST_MaxNURBSPointArrayElements* is the implementation-defined maximum cardinality of an array of *ST_NURBSPoint* values.

26) *ST_MaxOrganizationNameLength* is the implementation-defined maximum length used for the character representation of an organization name.

27) *ST_MaxPositionExpArrayElements* is the implementation-defined maximum cardinality of an array of *ST_PositionExp* values.

28) *ST_MaxReferentArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Referent* values.

29) *ST_MaxSRSAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text representation of an *ST_SpatialRefSys* value.

30) *ST_MaxSRSDefinitionLength* is the implementation-defined maximum length for the well-known text representation of a spatial reference system.

31) *ST_MaxStartValueArrayElements* is the implementation-defined maximum cardinality of an array of *ST_StartValue* values.

32) *ST_MaxSRSNameLength* is the implementation-defined maximum length used for the character representation of the identifier of a spatial reference system.

33) *ST_MaxTopologyName* is the implementation-defined maximum length of the CHARACTER VARYING topology name.

34) *ST_MaxTopologyOrNetworkName* is the implementation-defined maximum length of the CHARACTER VARYING topology or network name.

35) *ST_MaxTypeNameLength* is the implementation-defined maximum length used for the character string representation of a type name.

36) *ST_MaxUnitNameLength* is the implementation-defined maximum length used for the character representation of a unit indication.

37) *ST_MaxUnitTypeLength* is the implementation-defined maximum length used for the character representation of the type of a unit of measure.

38) *ST_MaxVariableNameLength* is the implementation-defined maximum length used for the character representation of an implementation-defined meta-variable.

39) *ST_MaxVectorArrayElements* is the implementation-defined maximum cardinality of an array of *ST_Vector* values.

40) *ST_MaxVectorAsBinary* is the implementation-defined maximum length of the BINARY LARGE OBJECT used for the well-known binary representation of an *ST_Vector* value.

41) *ST_MaxVectorAsGML* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the GML representation of an *ST_Vector* value.

42) *ST_MaxVectorAsText* is the implementation-defined maximum length of the CHARACTER LARGE OBJECT used for the well-known text and GML representation of an *ST_Vector* value.

43) *ST_TypeCatalogName* is the implementation-defined character representation of the name of the catalog, which contains the descriptor of the data type *ST_Geometry*.

44) *ST_TypeSchemaName* is the implementation-defined character representation of the name of the schema, which contains the descriptor of the data type *ST_Geometry*.

Annex B

(informative)

Implementation-dependent elements

This Annex references those places where this part of ISO/IEC 13249 states explicitly that the actions of a conforming implementation are implementation-dependent.

The term implementation-dependent is used to identify characteristics that may differ between implementations, but that are not necessarily specified for any particular implementation.

1) Subclause 8.3.9, "ST_BdPolyFromText Functions"

   a) Description 4) d)

      Using an implementation-dependent algorithm, an exterior linear ring, *ELR*, and an array of zero or more interior rings, *AILR*, are determined from the array of linear rings in *AMLS*.

2) Subclause 8.3.10, "ST_BdPolyFromWKB Functions"

   a) Description 4) d)

      Using an implementation-dependent algorithm, an exterior linear ring, *ELR*, and an array of zero or more interior rings, *AILR*, are determined from the array of linear rings in *AMLS*.

3) Subclause 9.6.7, "ST_BdMPolyFromText Functions"

   a) Description 4) d)

      Using an implementation-dependent algorithm, an array of *ST_Polygon* values, *APA*, is determined from the array of linear rings in *AMLS*.

4) Subclause 9.6.8, "ST_BdMPolyFromWKB Functions"

   a) Description 4) d)

      Using an implementation-dependent algorithm, an array of *ST_Polygon* values, *APA*, is determined from the array of linear rings in *AMLS*.

5) Subclause 11.3.15, "ST_RemEdgeModFace Procedure"

   a) Description 2) e) iv)

      If neither *F1* nor *F2* is equal to 0 (zero), then the choice of which face to modify and which to delete is implementation-dependent.

6) Subclause 13.1.1, "ST_SpatialRefSys Type"

   a) Description 2)

      The attribute definitions in the *ST_SpatialRefSys* type are implementation-dependent.

   b) Description 2) NOTE)

      Implementations should refer to ISO 19111 as a model to follow for the implementation-dependent attribute definitions in the *ST_SpatialRefSys* type.

7) Subclause 13.1.5, "ST_SRID Method"

   a) Description 2)

      A spatial reference system identifier that is not equal to 0 (zero) is implementation-dependent.

Annex C

(informative)

Deprecated features

It is intended that the following features will be removed at a later date from a revised version of this part of ISO/IEC 13249:

1) The use of the coord XML element.
2) The use of the coordinates XML element.
3) The use of the following <uint32> Values from Table 15 — <well-known binary representation> <uint32> Values: 1000001, 1000002, 1000003, 1000004, and 1000005.
4) The use of the ST_CircularString type for circles.

Annex D

(informative)

Incompatibilities with ISO/IEC 13249-3:2011

This edition of this part of ISO/IEC 13249 introduces some incompatibilities with the earlier version of Information technology — Database languages — SQL multimedia and application packages – Part 3: Spatial as specified in ISO/IEC 13249-3:2011.

Except as specified in this Annex, features and capabilities of Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial are compatible with ISO/IEC 13249-3:2006.

1) In ISO/IEC 13249-3:2011, ST_CompoundCurve could only be made up of ST_LineString and ST_CircularString values. In ISO/IEC 13249-3:201x , values of all subtypes of ST_Curve are allowed, including newly added curve types and ST_CompoundCurve in order to become consistent with ISO 19107 and GML.
2) In ISO/IEC 13249-3:2011, ST_CircularCurve could have equal segment start and end points, resulting in a (non-deterministic) circle. In ISO/IEC 13249-3: 201x, this is no longer permitted. A new type, ST_Circle, is introduced, similar to ISO 19107 and GML.

Annex E

(informative)

Geometry Type Hierarchy

The subtype relationships between geometry types are shown in Figure E.1 — Geometry Type Hierarchy Diagram.

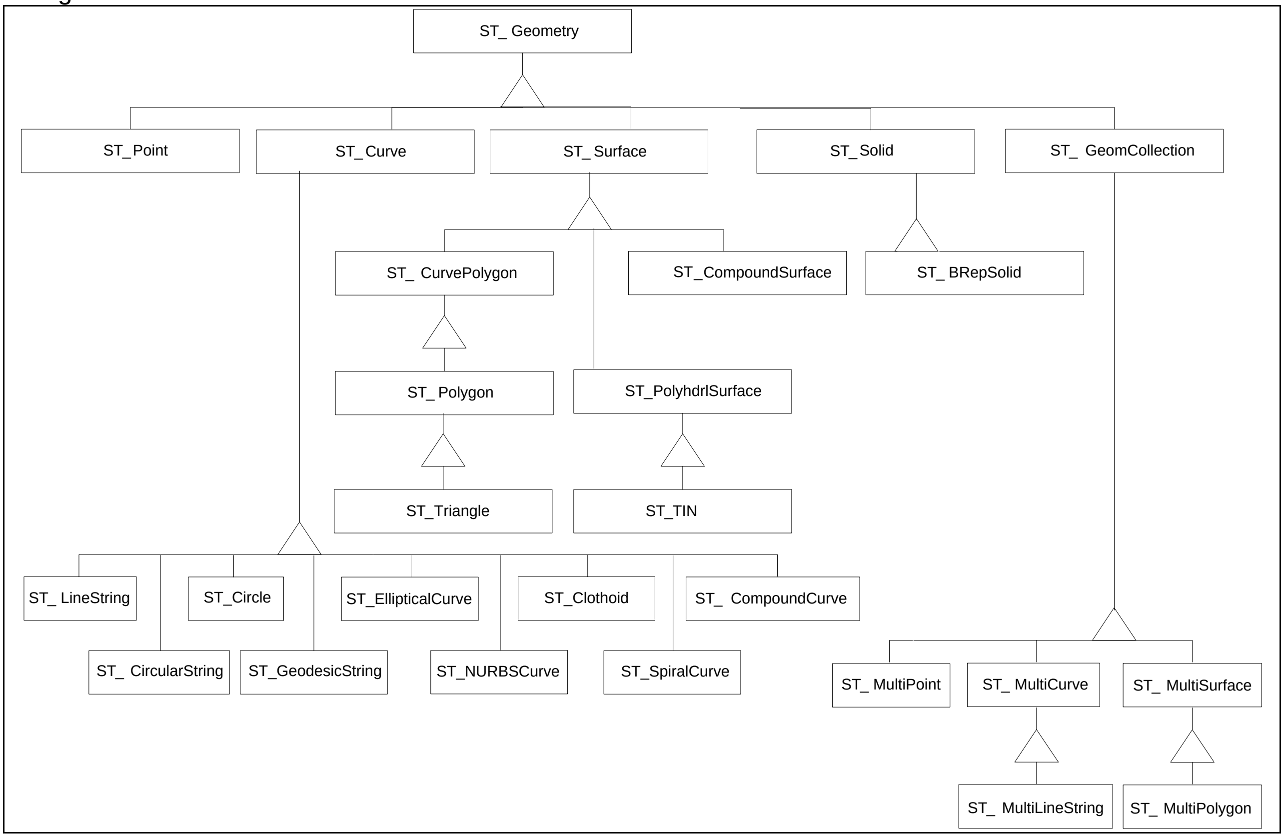

Figure E.1 — Geometry Type Hierarchy Diagram

In Figure E.1, each box identifies a user-defined type. The line with a triangle symbol connecting an upper box with one or more lower boxes indicates the user-defined types in the lower boxes are direct subtypes of the user-defined type in the upper box.

The following geometry types are supported: ST_Geometry, ST_Point, ST_Curve, ST_LineString, ST_CircularString, ST_Circle, ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve, ST_CompoundCurve, ST_Surface, ST_CurvePolygon, ST_Polygon, ST_Triangle, ST_PolyhdrlSurface, ST_TIN, ST_CompoundSurface, ST_Solid, ST_BRepSolid, ST_GeomCollection, ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface, and ST_MultiPolygon.

ST_Geometry has the following subtypes: ST_Point, ST_Curve, ST_Surface, ST_Solid, and ST_GeomCollection.

ST_Curve has the following subtypes: ST_LineString, ST_CircularString, ST_Circle, ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve, and ST_CompoundCurve.

ST_Surface has the following subtypes: ST_CurvePolygon, ST_PolyhdrlSurface and ST_CompoundSurface

ST_CurvePolygon has the following subtype: ST_Polygon.

ST_Polygon has the following subtype: ST_Triangle.

ST_PolyhdrlSurface has the following subtype: ST_TIN.

ST_Solid has the following subtype: ST_BRepSolid.

ST_GeomCollection has the following subtypes: ST_MultiPoint, ST_MultiCurve, and ST_MultiSurface.

ST_MultiCurve has the following subtype: ST_MultiLineString.

ST_MultiSurface has the following subtype: ST_MultiPolygon.

Bibliography

[1] *OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option, OGC-06-104r3*, Open Geospatial Consortium, Inc., October 5, 2006.

[2] *OGC® Geography Markup Language (GML) — Extended schemas and encoding rules*, OGC 10-129r1, Open Geospatial Consortium, Inc., February 7, 2012.

[3] de Berg, Mark, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf, *Computational Geometry, Algorithms and Applications,* Springer-Verlag, 1997.

[4] Scarponcini, Paul, Generalized Model for Linear Referencing in Transportation, *GeoInformatica*, 6(1): 35-55, 2002.

[5] ISO IS 19109:2005, *Geographic information — Rules for applications schema*, 2005.

Index

An index entry appearing in **boldface** indicates a range of pages where a user-defined type is defined in this part of ISO/IEC 13249. All other index entries appear in roman type.

An index page number appearing in **boldface** indicates a page or range of pages where the attribute, routine, or user-defined type is specified. An index page number appearing in *italics* indicates a page where the word, phrase, attribute, routine, or user-defined type is defined. An index page number appearing in roman type indicates a page where the word, phrase, attribute, routine, or type was used.

—<—

—0—

—1—

—2—

—3—

—A—

—B—

—C—

—D—

—E—

—F—

—G—

—H—

—I—

—K—

—L—

—M—

—'—

—M—

—N—

—T—

—U—

—V—

—W—

—X—

—Y—

—Z—

